Institut für Nachrichtenvermittlung und Datenverarbeitung (IND) der Universität Stuttgart

Prof. Dr.-Ing. Dr. h.c. Paul Kühn

Verwendung von Flows zur Analyse und Messung von Internet–Verkehr

Diplomarbeit

von

Siegfried Löffler

<siegfried.loeffler@rus.uni-stuttgart.de>

Betreuer:	Martin Lorang
Beginn:	Dezember 1996
Ende:	August 1997

Institute of Communication Networks and Computer
Engineering (IND) of the University of Stuttgart

Using Flows for Analysis and Measurement of Internet Traffic

Diploma Thesis

by

Siegfried Löffler

<siegfried.loeffler@rus.uni-stuttgart.de>

Abstract

The term “flow” is being used in at least three different contexts in the Internet environment: It is used to describe traffic for resource reservation protocols like RSVP. “Flows” are also considered as a unit for traffic switching. Finally, flows are a rather new category in network measurement and analysis.

In this work, the focus is on using flows for traffic measurement and analysis.

This field has become important because of the rapid growth of the Internet and the growing demand for multi–media applications which require high bandwith network resources. New tools have to be developed for the analysis and measurement of traffic at high line speeds. These tools have to provide the information necessary for network planning and configuration, resolution of congestion problems as well as for user accounting and charging.

This report describes the use of a flow based methodology [14] as a means to analyze and monitor traffic. Measurement applications employing flow methodologies are compared. Following the trend to integrate network management technologies into a World–Wide–Web framework, a Java based traffic analyser is presented as a contribution to the IETF Realtime Traffic Flow Measurement (RTFM) architecture [4, 5, 26]. The developed applet uses the “Simple Network Management Protocol” (SNMP) for communication with RTFM traffic flow meters. It allows network managers to obtain flow–based network status information in real–time using a standard web browser.

Contents

List of Figures

List of Tables

Table of Abbreviations

100VG–AnyLan	100 Mbit/s Network Technology by Hewlett–Packard
AAL	ATM Adaption Layer
AS	Autonomous System
ASN.1	Abstract Syntax Notation One
ATM	Asynchronous Transfer Mode
BER	Basic Encoding Rules
CGI	Common Gateway Interface
CMIP	Common Management Information Protocol
CPCS	Common Part Convergence Sublayer
CPU	Central Processing Unit
FDDI	Fiber Distributed Data Interface
FIN	Finish Flag (TCP header)
FTP	File Transfer Protocol
HTTP	Hypertext Transfer Protocol
IETF	Internet Engineering Task Force
IFMP	Ipsilon Flow Management Protocol (RFC1953)
IOS	Router/Switch–Betriebssystem von Cisco
IP	Internet Protocol
IPv4	Internet Protocol Version 4 (standard IP)
IPng/IPv6	Internet Protocol Next Generation / Version 6
ISDN	Integrated Services Digital Network
MIB	Management Information Base
NeTraMet	Network Traffic Meter

NeMaC	NeTraMet Manager / Collector
MPOA	Multiprotocol over ATM (ATM Forum)
NLANR	National Laboratory for Applied Network Research
OC3	ATM Optical Carrier 3 (150 Mbit/s ATM)
OID	Object Identifier
OSI	Open Systems Interconnection
PDU	Protocol Data Unit
PTI	Payload Type Identifier
PVC	Permanent Virtual Circuit
QoS	Quality of Service
RFC	Request For Comments
RSVP	Resource Reservation Protocol
RUS	Rechenzentrum der Universität Stuttgart
RMON	Remote Monitoring (Standard for)
RTFM	Realtime Traffic Flow Measurement (Working Group within the IETF)
SAR	Segmentation and Reassembly
SAS	Secure (SNMP) Applet Server
SDU	Service Data Unit
SNMP	Simple Network Management Protocol
SNMPv2	Simple Network Management Protcol Version 2
SVC	Switched Virtual Circuit
SYN	Synchronize Sequence Numbers Flag (TCP Header)
TCP	Transmission Control Protocol
TCP/IP	Transmission Control Protocol / Internet Protocol
UDP	User Datagram Protocol
vBNS	Very High Speed Backbone Network System
VBR	Variable Bit Rate
VCI	Virtual Channel Identifier
VM	Virtual Machine
VPI	Virtual Path Identifier

WBEM	Web Based Enterprise Management
WWW	World Wide Web

Chapter 1

Introduction

The term “flow” is being used in at least three different contexts in the Internet environment: First, it is used to describe traffic for resource reservation protocols like RSVP. Second, “Flows” are also considered as a unit for traffic switching. Finally, flows are a being used as a rather new category in network measurement and analysis.

In this work, the focus is on using flows for traffic measurement and analysis.

Before introducing flows in chapter 2, as a motivation we give an overview of typical network management tools. We also describe the use of the Web technology as an up–to–date framework for such applications.

1.1 The Need for Traffic Analysis

During the last years, the Internet has experienced a rapid growth. New multimedia applications demand for increased bandwidth. The World–Wide–Web with its intuitive interface has brought the Internet close to the masses and the number of users has therefore dramatically increased. Because of this, faster and faster backbone networks (like for example the vBNS[1]) are being deployed. But not only the backbone network speed is rising. For multimedia applications, the traditional Ethernet is more and more often replaced by network technologies like 100–base–T (also known as “Fast Ethernet”), 100VG–AnyLan, ATM, FDDI etc. For traffic analysis, measurement and accounting on these high speed networks new applications are needed that can cope with the increased traffic.

1.1.1 Traffic Analysis and Network Monitoring

Problems arise for example with *traffic analysis*, especially when it comes to the *debugging of a faulty network*. On standard Ethernet, most network administrators use the “`tcpdump`” tool written by Van Jacobson [2] to locate machines that transmit excessive data or to debug why one host is not able to communicate with another. Although it is possible to set filters for the “`libpcap`”[3] packet capturing library used by `tcpdump` the tool needs a fast machine and generates a high CPU and bus load. It makes use of a special “promiscuous” mode for the network adapter. In this mode, every received packet is passed to the networking software, not depending on whether it was addressed to the machine or not. The resulting interrupt[4] and CPU load can get very high with fast line speeds. An ATM OC3 connection — which is more or less the standard for desktop ATM connections today — has a transfer rate of 150 Mbit/s. This is far too much for a standard PC or workstation to monitor in realtime with `tcpdump`. New tools and methods are needed to allow traffic monitoring and analysis on such high speed links.

An additional difficulty gets more and more important when bigger amounts of data have to be analyzed: In order to resolve a problem, the administrator has to have an idea of what he is searching for in advance. This makes the debugging difficult. Tools that can for example *immediately identify traffic sources that produce excess*

[1] vBNS = very High Speed Backbone Network Service, see `http://www.vbns.net`

[2] The source code for `tcpdump` is available at `http://www-nrg.ee.lbl.gov/`

[3] The `libpcap` library is available from `ftp://ftp.ee.lbl.gov/libpcap-*.tar.gz`.

[4] On UNIX systems, usually a hardware interrupt is generated for each received packet

traffic and that can give an *instant overview over the traffic state* on a given link would be helpful here.

Traffic monitoring is an important field as well. Monitoring is not only useful to get information about the kinds of applications that are used on the network, it is essential for security measurements. One of the most important steps when setting up a secure environment is the installation of a monitoring system. This system can for example be used to trace back the path of an intruder that is being found on a system or it can be used to get information about attacks as early as possible.

1.1.2 Planning of Infrastructural Development

Another field in which network analysis is important is the planning of the infrastructure. A network administrator needs to know how loaded the network backbone is, and the more exact information he has about the nature of the traffic on his network, the better he can determine what he will have to improve next. For this reason, it is important to have as detailed measurement information as possible. Conventionally, the network administrator would just check counters (for example using SNMP tools like HP OpenView, shown in Figure 1.1, or possibly RMON probes) for his link or maybe just watch collision LEDs to determine how loaded the link is. Once he decides that there are too much collisions, he will try to replace the medium by a faster one or split it into multiple segments. However, this does not allow him at all to determine the applications that are responsible for the traffic growth. If for example the traffic growth on a LAN would be only due to an increased number of people surfing the World Wide Web, the administrator would eventually better add a new proxy server for this LAN segment than just split it into two segments.

Not less important is to do *continued measurements over longer periods of time*. Doing this allows the network adminstrator to get early insights in trends for new protocols being used. New kinds of applications often result in new needs for the networking equipment. Therefore it is important for the administrator to recognize those trends as soon as possible.

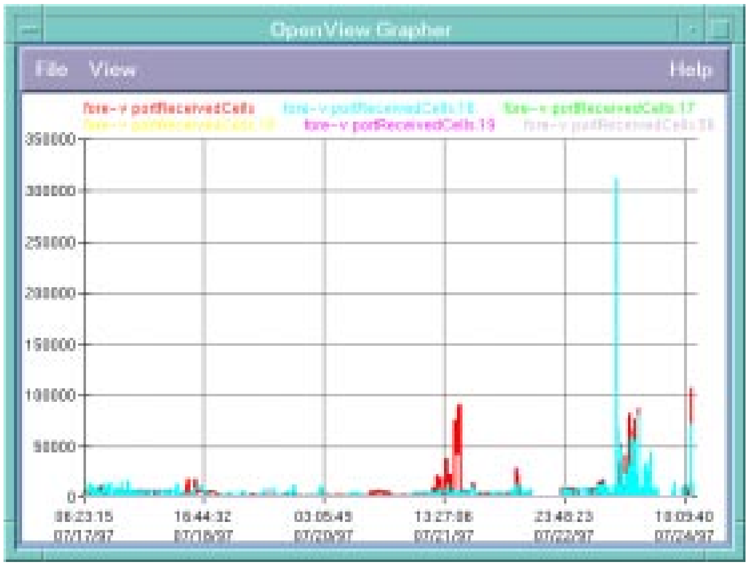


Figure 1.1: SNMP Counter Graph generated with HP OpenView

1.1.3 Traffic Measurement and Accounting

The *commercialization of the Internet* has not only resulted in an increase of the number of users. Probably the most important problem that arises is the *accounting* of the transferred data. Traditionally, the Internet was a network between research and educational institutions. The connected institutions usually payed a fixed fee for their connection. Nowadays, Internet providers would like to charge their clients depending on the volume of data they transfer. To do this, they need powerful tools that are able to count the transferred amount of data. The higher the line speeds are, the more difficult this is.

It is still a common practice for providers to charge fixed monthly fees for Internet access. Often the only reason for this is that they have no means to do an exact accounting for all clients.

Traditional solutions for volume based traffic charging include:

1. **Reading the SNMP Octet Counters from the Routers**

Most if not all modern networking equipment offers the possibility to gather statistics about the amount of data that was transferred via its interfaces. For this purposes, usually counters for bytes and/or packets that are transferred over each interface are provided. These counters can be queried using the SNMP procotol.

The disadvantage of this is that only the total amount of traffic transferred can be accounted. It is not possible to apply different prices depending on the kind of traffic. Additionally, since the SNMP counters are only maintained once for each hardware port, a separate port is necessary for each client that is to be accounted. These additional expenses for hardware make this solution unattractive.

2. **Using RMON / RMON2 probes**

   The RMON standard, which is described in RFC1757 [39], was designed to provide proactive monitoring and diagnostics for distributed LAN–based networks. Special monitoring devices, called agents or probes, allow the monitoring of critical network segments and to set off user–defined alarms. RMON has been implemented in special stand–alone hardware, embedded in switches and as a program running on PCs or workstations. Communication with the probe is implemented using the SNMP protocol.

   In theory, the RMON standard would be suitable for higher line speeds. It has however shown that it is difficult to adapt RMON to protocols like 100VG–AnyLan or ATM. No RMON implementations for those protocols are available at this time. For ATM, a first attempt was AMON (ATM Circuit Steering MIB), which defines a way to copy traffic from a virtual circuit (VC) to a location where an external probe can decode it. AMON — which was proposed by Fore — has been discussed in the ATM Forum since summer 1995. However, progress has been so slow that the forum threatened to suspend the AMON MIB group’s work. In march 1996 Cisco — although one of the founder members of the ATM Forum — has surprised the networking community by submitting a draft for an “ATM RMON MIB” to the IETF rather than to the ATM Forum. Cisco has developed the ATM RMON MIB without discussing it with other manufacturers of ATM hardware. This unusual way of presenting their proposal has been the reason for controversial discussions. It is therefore not very likely that their proposal will become a standard in the near future.

3. **Using a PC or Workstation running** `tcpdump`
   The `tcpdump` tool mentioned above can be used to monitor all traffic that passes through a network adapter in a PC or workstation. Using it permits as well to count the data that is received by this network adapter. However as mentioned above the interrupt load using `tcpdump` is a problem when the data is being received at higher line speeds. When the load is getting too high, the probability for packet losses is growing. This technique is used at a local Internet provider in Stuttgart[5], and it was found that even on transfer rates of about 10 Mbit/s (standard ethernet) there is already a probability for packet loss in the range of 1%. Obviously this solution is only practicable for lower line speeds. It nevertheless offers maximum flexibility since a user–written program can be used to analyze a trace of all the headers from the packets the machine receives.

[5] Please contact the author for details about that installation

1.2 The WWW as a Framework for Network Management

A trend that has just begun on the Internet is to integrate all kind of information systems into WWW or intranet environments. This is also interesting for network management and monitoring. The WWW technology offers an ideal user–interface for management applications. Data that is put on a web server immediately is accessible from everywhere on the network, without having to install any additional software on a lot of machines.

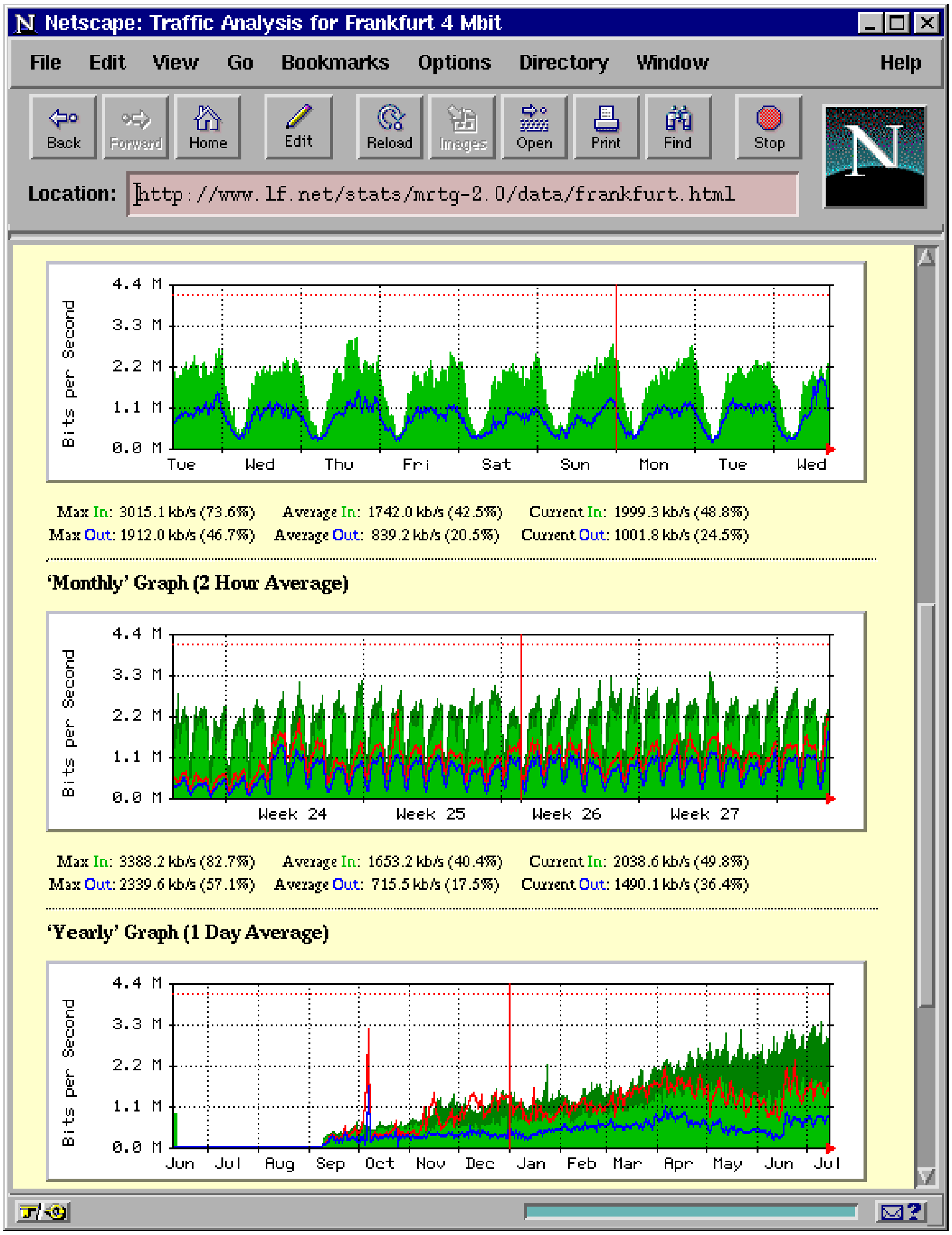
Recently, a broad variety of products for web based network management has been announced[6]. The range goes from solutions that provide graphs for SNMP counters to complete management environments that allow the modification and status display of manageable workstations, routers etc.

1.2.1 Traffic Graphing with "mrtg"

One example for a web–based network monitoring solution is "`mrtg`" (Multi Router Traffic Grapher), a rather small program written in Perl and C [19, 21] by Tobias Oetiker. The author has made this program available within the public domain, and therefore it is already widely in use. It can partly replace functionality for which it was earlier necessary to buy expensive commercial network management solutions.

The program itself consists of two parts. A first program is called in regular intervals and queries a set of SNMP variables. Those values are stored in a file on the web server. This file is then processed by a second program and a web page with graphs depicting the counters evolution is generated. Figure 1.2 shows a screenshot of such a web page generated with "`mrtg`". The program is easy to install and highly configurable for use with different networking equipment. Since it is available as source code, it is also quite easy to extend.

[6] See `http://www.mindspring.com/~jlindsay/webbased.html`



Location: http://www.lf.net/stats/mrtg-2.0/data/frankfurt.html

Max In: 3015.1 kb/s (73.6%) Average In: 1742.0 kb/s (42.5%) Current In: 1999.3 kb/s (48.8%)
Max Out: 1912.0 kb/s (46.7%) Average Out: 839.2 kb/s (20.5%) Current Out: 1001.8 kb/s (24.5%)

‘Monthly’ Graph (2 Hour Average)

Max In: 3388.2 kb/s (82.7%) Average In: 1653.2 kb/s (40.4%) Current In: 2038.6 kb/s (49.8%)
Max Out: 2339.6 kb/s (57.1%) Average Out: 715.5 kb/s (17.5%) Current Out: 1490.1 kb/s (36.4%)

‘Yearly’ Graph (1 Day Average)

Figure 1.2: Web based traffic analysis with `mrtg`

1.2.2 Network Management with "IntraSpection"

An example for a complete management solution is "IntraSpection" by Asanté Technologies[7]. The product is also usable for the generation of graphs of SNMP variables, however its design is quite different from "`mrtg`". It is not designed for a long–term analysis of traffic counters but to provide instant access to management information. One main functionality is the so called "network map discovery". Figure 1.3 shows how "IntraSpection" can be used to display an overview of all SNMP capable networked equipment in an IP subnet. The program can be used to browse the available information on the hosts as well as to produce graphs over counters. Those graphs are generated in realtime using a Java applet, as shown in Figure 1.4. Another important feature that comes with "IntraSpection" is the SNMP trap management which can be used to notify the network adminstrator of events he has to react on immediately.

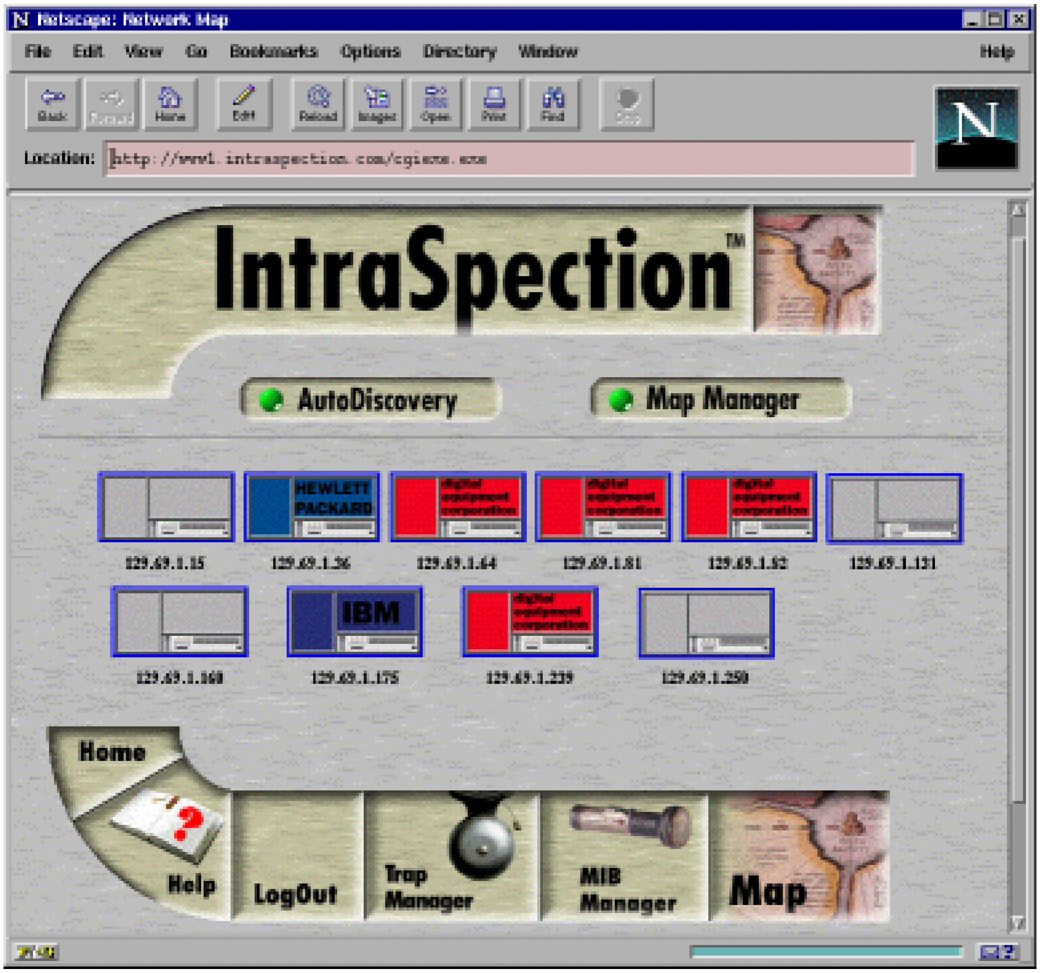
1.2.3 Web Based Enterprise Management

Currently, many companies including Cisco, Compaq, Intel and Microsoft are working on an interesting framework. "*Web–based Enterprise Management*" (WBEM)[8] is one of the main reasons why web based network management products like the one we just saw, will probably become important in the near future.

The key purpose of the WBEM initiative is to consolidate and unify the data provided by existing management technologies. The focus is on solving real enterprise issues by allowing problem areas to be tracked from end to end — from the user/application level through the systems and network layers to remote service/server instances. However, WBEM does not attempt to replace existing management standards such as SNMP or CMIP, but it provides a framework into which those techniques fit.

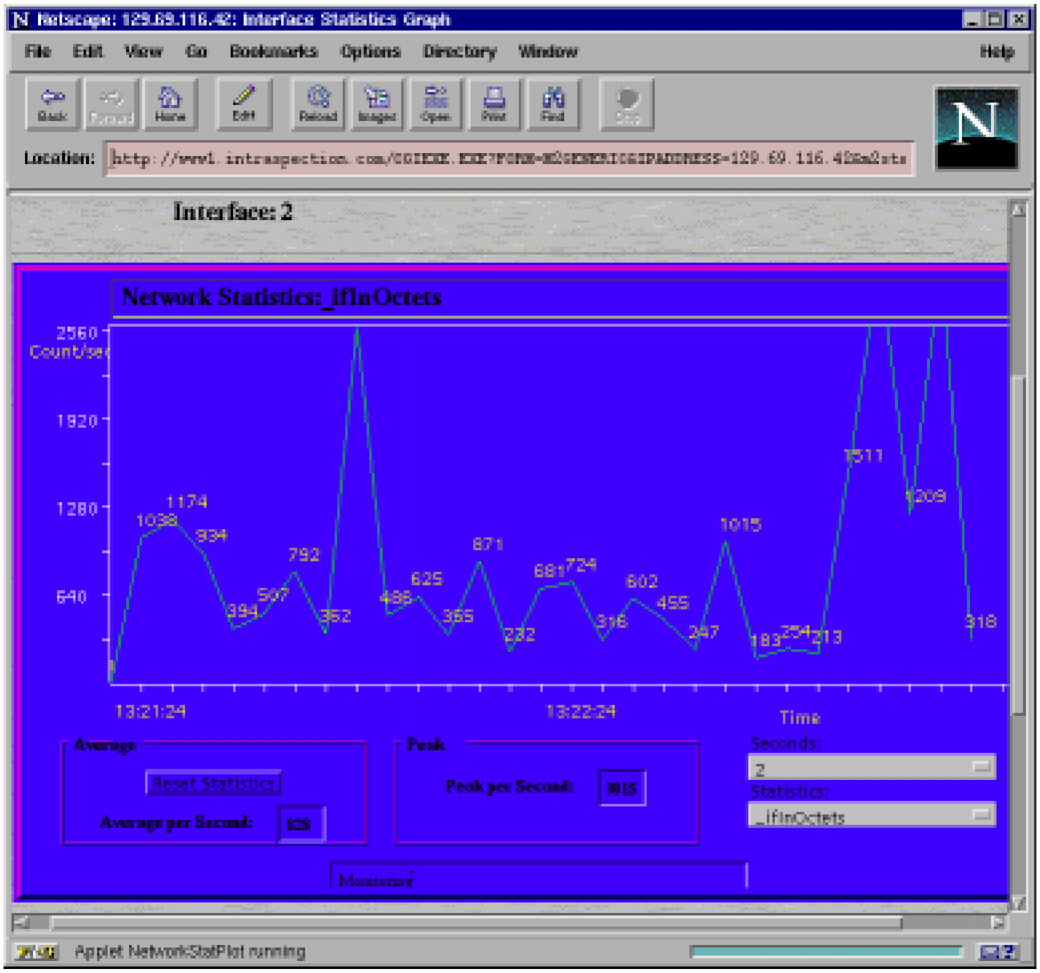
[7] A demo version of IntraSpection is on the WWW at `http://www.intraspection.com`

[8] A web page is provided by the WBEM initiative at `http://wbem.freerange.com`



Location: http://www1.intraspection.com/cgiexe.exe

Figure 1.3: Network management with Intraspection: Map Discovery



Netscape: 129.69.116.42: Interface Statistics Graph

Location: http://www1.intraspection.com/CGIEXE.EXE?FORM=M2GENERIC&IPADDRESS=129.69.116.42&m2sta

Interface: 2

Network Statistics: _ifInOctets

Average per Second: 528

Peak per Second: 3015

Seconds: 2

Statistics: _ifInOctets

Monitoring

Applet NetworkStatPlot running

Figure 1.4: Network management with Intraspection: Counter Graphing in Java

The networking hardware will be only one part of the managed objects in a networked enterprise. Obviously it will be easy to integrate existing web based network management applications in such a framework, therefore the effort seems to be a good investment in the future.

Chapter 2

Flows

2.1 Flow Definitions

2.1.1 Historical Definitions of Flows

The Packet Train Model

One of the first models of a traffic flow was created by Jain [11] in his *packet train* model. He defines a *packet train* as a burst of packets arriving from the same source and heading to the same destination. If the spacing between two packets exceeds some inter–train gap, they are said to belong to different trains. In his model, the inter–train time is a user parameter, dependent on the frequence with which applications use the network. The inter–car arrival for a train is a system parameter and depends on the network hardware and software.

This model reflects the fact that much of the communication inside a network involves in fact many packets spaced closely in time that are exchanged between the same two endpoints.

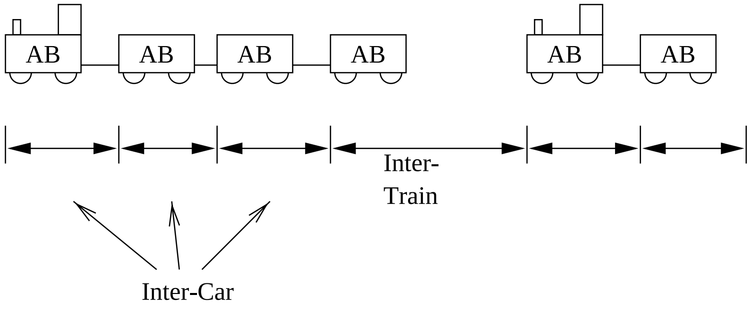


Figure 2.1: Times in the Packet Train Model

Defining Flows based on TCP Connections

The first efforts to define a *packet train* or traffic flow on the network focused on connections. When using the TCP protocol, all connections are handled via the SYN and FIN control mechanism. It is therefore possible to watch the traffic on a network, check for SYN and FIN packets and thereby aggregate everything with identical service number, source and destination address etc between the SYN and FIN packet into one “flow” [25]. The strength of this approach is that the detection of beginning and end of a TCP connection based flow is relatively easy.

2.1.2 Motiviation For a New Flow Definition

The practical use of such connection based flows is however restricted. Although the theory exists already for a while, it has never really been used in implementations. This is due to some problems which are related to the nature of the Internet environment:

1. The Internet being a *connectionless datagram environment*, dependence on connection–oriented information will often interfere with operational stability. If routes change during a flow, new routers will carry datagrams that never saw the transport layer SYN/SYN–ACK packets, and routers that did

see earlier datagrams in a flow will never see the FIN/FIN–ACKs. Flow state information that is dependent on this data will become obsolete and never expired in such cases.

2. *IP Fragmentation* would also pose problems since all but the first fragment lack the TCP/UDP port information and therefore it would be impossible to track the fragmenented parts of a packet into a higher layer flow.

3. Not all traffic makes use of connection oriented transport layer protocols. The trend on the Internet is that more and more lightweight protocols are used that do not use the TCP mechanism for connection setup and teardown. Especially new multimedia applications usually bring their own concepts for connection handling.

4. Finally, new link level technologies, e.g. ATM, will not have access to transport layer informations; any Internet related transmission decisions will have to rely only on IP level information. In particular, until end–to–end ATM is a reality, IP gateways attached to ATM style networks will have to multiplex possibly many IP flows onto ATM. Mapping higher level (IP) flows to underlying link level virtual circuits (VCs) will require effective setup, maintenance and timeout strategies as well as accounting schemes.

Having seen the efforts to extend the *packet train* model of flows to the transport or application layers [1, 22, 25] or focusing on TCP traffic flows [9, 31], Claffy, Braun and Polyzos have introduced a more generalized, comprehensive methodology of a timeout–based flow characterization on the IP layer [14]. Their flow definition is also based on the packet–train model, but in contrary to the other models mentioned they avoid to use connection information.

The IP layer, the “heart” of internet technology, is a connectionless network layer. Not to use connections for routing or switching was one of the main reasons why the Internet could grow as rapidly as it did. Any connection oriented service on the Internet is implemented in the transport protocol that is running on the end hosts only. Routers on the Internet are simple and fast since they just rely on the IP headers of the packets. This has shown to scale very well. In the same way, the connectionless techniques are scaleable for measurement and analysis applications.

The classification of networks into either connectionless or connection oriented ones is pretty restrictive. In particular it is obviously not true that in a connectionless network all datagrams are completely independent. The datagrams are certainly switched independently, but it is usually the case that a stream of datagrams between a particular pair of hosts flow through a particular set of routers. Hence the idea to define a *flow* as *a sequence of packets* matching the same criteria is a useful concept for an abstraction.

By using this abstraction of connectionless traffic flows, a set of new applications will become possible. First of all, we can use the flows for network monitoring, measurement and analysis, as primarily shown in this report. However, they are also interesting for routing, switching, as we will see in section 2.5 as well as for congestion control ([20], pp. 395 ff).

2.2 A Flow Model that is Suitable for the Internet

The flow model described by Claffy, Braun and Polyzos [14] defines a flow from a rather abstract point of view: A flow is **a sequence of packets matching certain criteria, exchanged between two entities on a network.** An example for such a flow could be all packets travelling through a certain point of a network that have identical source and destination IP network addresses.

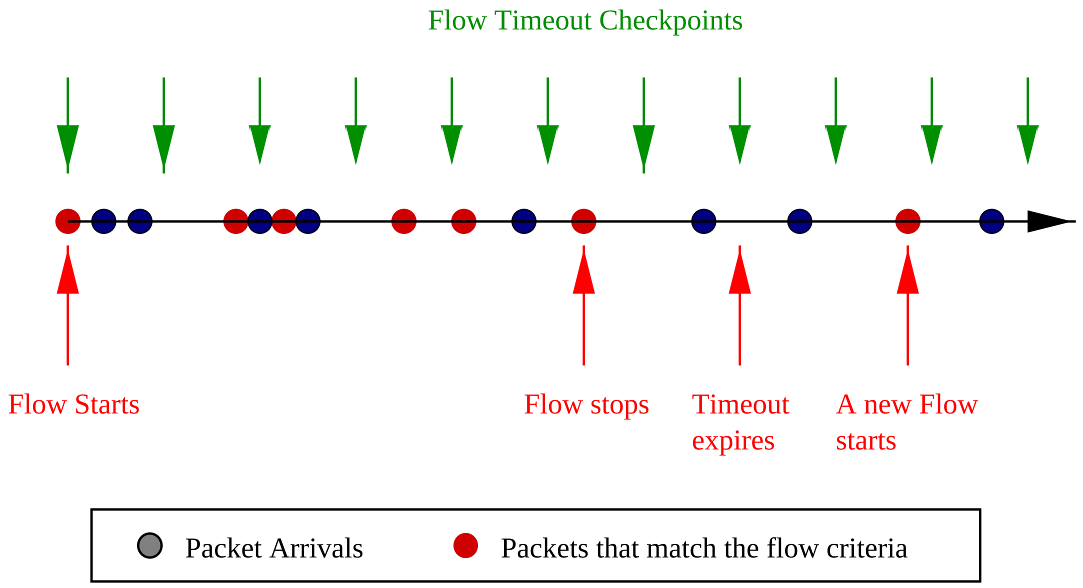


Figure 2.2: Defining a flow based on a timeout during idle periods

Figure 2.2 shows the timings that are relevant for this definition of a flow. The circles on the time axis depict the arrival of data packets. The data packets which match the flow criteria are shown as red circles. The green arrows mark the "flow timeout checkpoints". At those points we check whether information that matches the *flow criteria* (or *flow specification*) has been received since the last checkpoint. If information was received, the flow is called "*current*" or "*active*". If no information for a flow was received during the interval, the flow timeout expires. The flow is then called "*inactive*". Any new packets that match the same flow criteria will then belong to a new flow.

Two parameters in this model have to be further investigated. The first is the *flow specification*, which is essential for the classification of arriving packets into flows, is to be examined. The second is the value of the *flow timeout*, which can be varied. It is interesting to see the influence this variation has on flow measurements. In the following sections, we will investigate those two parameters in detail.

2.3 Flow Specifications

2.3.1 What are Flow Specifications?

In the previous section, we described the basic idea behind a traffic flow, which is to aggregate traffic that *matches certain criteria.* What we have not spoken about yet is how those criteria can be defined.

The most important advantage of our very general definition is that the communicating entities are not yet specified any further. Until now, we only assumed that they are two machines on the network that communicate with each other. However, the entities do not have to correlate with machines (i.e. network addresses) but they can as well be whole subnets of machines, classes of applications or autonomous systems [1]. In fact, anything that can be used to distinguish data packets is a potential criteria. In the following sections we will further specify which criteria can be used for the so called "*flow specification*".

A first proposal for a *flow specification* was given by Partridge in RFC1363 [29]. This proposal is however mainly motivated by the ideas of resource reservation functionality. Therefore, Partridges *flow specification* is focused on Quality of Service (QoS) parameters which shall be used for the reservation of bandwidth for certain kinds of multimedia traffic. The *flow specification* he proposes is to be used by the network for admission control and resource allocation purposes.

For network measurement and analysis — our main interest — we need a different approach. Our aim is to aggregate as much information as possible about the high amount of data that is transferred and at the same time to use as less memory for this aggregation as needed. Since we usually do not know in advance what we are searching for and what we want to measure, we need a flexible way to define what kinds of traffic we want to aggregate.

[1] An autonomous system (AS) is an organizational entity of networks/machines

2.3.2 Flow Directionality

First, one can define a flow as *unidirectional* or *bidirectional*. While TCP traffic always is connection oriented and therefore always must be bidirectional, it still often exhibits strong asymmetries in the traffic profile of the two directions. Each TCP flow from A to B also generates a reverse flow from B to A, at the very least for small acknowledgment packets.

For data aggregation, we may or may not be interested in measuring those two flows separately, therefore measurement and analysis applications should ideally be configurable in regard to this parameter[2].

In the Internet environment is is possible to use a unidirectional definition of flows, i.e., bidirectional traffic between A and B is to be seen as two separate flows: traffic from A to B, and traffic from B to A. This allows to get interesting insights for the analysis of routing issues or traffic characteristics. The aggregation of those two flows into one unidirectional flow could on the other hand be sufficient for accounting. Obviously it makes sense to allow the unidirectional defintion, since a later transformation of unidirectional flows into a bidirectional flow is always possible.

2.3.3 One vs. Two Endpoint Aggregations of Traffic

The second aspect of a flow is related to its endpoints. As mentioned the model allows us not only to examine data exchange between two entities on the network, it is as well possible to aggregate all traffic that originates from a specified entity or that is addressed to a specified entity. Such flow specifications are called "*single endpoint flows*" in contrary to the "*double endpoint flows*", where source and destination addresses are being specified.

An example where a single endpoint flow is interesting is the aggregation of all data transferred from a given destination network number. Those measurements could

[2] Claffy, Braun and Polyzos used unidirectional flows for their analysis in [14]

be compared to the traffic aggregated between this network number and a given second network number to calculate the percentile of traffic from the given network to another network.

2.3.4 Types of Flow Endpoints

An aspect already mentioned above, and certainly the most important criteria for *flow specifications* are the flow endpoints. The endpoint specification somehow has to describe the communicating entities. Potential granularities for this description include aspects such as traffic by

- **Hosts**, identified by
  - Network layer address (e.g. IP address)
  - Link layer address (e.g. ethernet address)
  - Symbolic hostname
- **Networks**, identified by
  - Network number
  - Domainname
- Abritrary **groups of hosts**
- Traffic sharing a **common path** on the network, identified by
  - Interface number on a backbone node
  - ATM connection identifiers (VCI/VPIs)

Various additional granularities could be defined, depending on the type of the local network installation and the demands of the user. The only common critera that all granularities have to fulfill is that it must be possible to check for each received packet whether it matches the critiera for the flow or not.

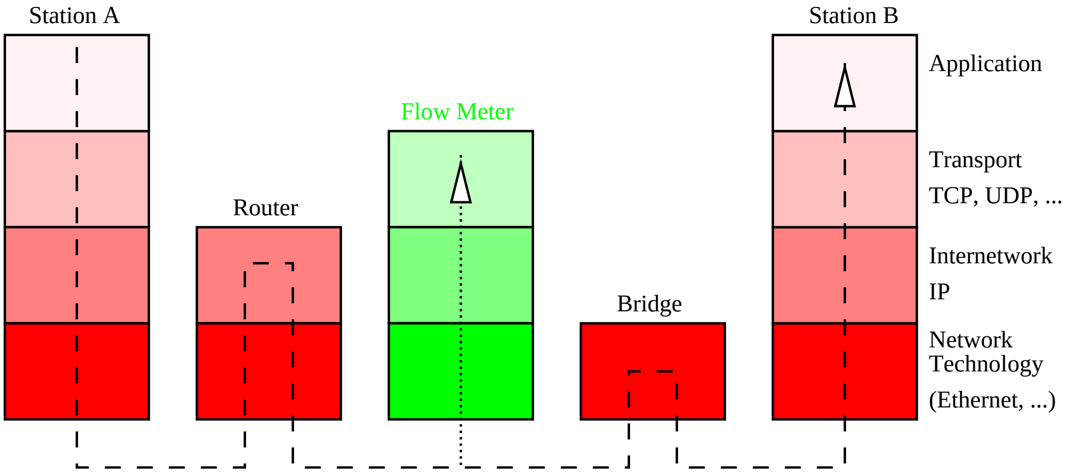


Figure 2.3: Flow–Measurement in the layered model of the Internet

Figure 2.3 illustrates where the flow endpoints could be positioned in the layered communication model of the Internet. If flows are to be specified with a granularity that reaches the application layer, the measuring entity will of course also have to have knowledge about the format of the data of this layer. In the TCP/IP model in order to define flows based on the transport layer, the port identifier field of the TCP header would have to be analyzed.

The granularities do not necessarily have an inherent order, as a single user or application might straddle several hosts or even several network numbers. Generally, flow criteria dont have to be restricted to single network layers. It is also possible to specify a flow using a *combination of different criteria* on several layers, for example one could aggregate all traffic that is generated by a specific application on a specific machine.

The possibility to define a flow in such a variable way is the huge advantage and strength of this model. When developing measurement applications, often it will show that one does not know exactly *what* should be measured in advance. The configurability of this model reflects this and by keeping as general and abstract as possible, the model is prepared to be usable for all kinds of analysis applications.

2.4 The Flow Timeout Parameter

As illustrated in Figure 2.2, a flow is "*current*" as long as incoming packets for it are not separated in time for more than the length of the interval between two timeout checkpoints. The length of this interval therefore is a parameter in this definition of flows that is worth further investigation.

In traditional measurements for traffic characterization, the times that are to be measured are usually ranging from nanoseconds to about one second. Many studies have for example focused on the mean inter–arrival time of packets. For measuring times in the range of micro– to nanoseconds, the granularity has to be very fine. In contrary, the time intervals we are talking about for flow measurements are rather macroscopic. Reasonable values for the flow–timeout are in the range of seconds to minutes.

Several people have worked on the timeout parameter. For their studies of wide–area traffic at the transport level, Caceres et al. [31] have used a 20 minute timeout, motivated by the FTP idle timeout value of 15 minutes. After comparing their results to a 5 minute timeout, they found only minimal differences for the number of established flows. Estrin and Mitzel [9] also compared timeouts of 5 and 15 minutes and found only little differences for the flow durations at those two values. Acharya and Bhalla [1] used a fixed 15 minute flow timeout.

In [6] and [14], Claffy, Braun and Polyzogos extensively examine Internet traffic on a large vBNS backbone node using different values for the flow timeout. One interesting result they found is that the majority of the flows between two hosts on the Internet don't even last longer than ten seconds. To examine the significance of the parameter, they used flow timeouts of 4, 32, 256 and 2048 seconds and compared the measurements.

Shorter timeouts tend to split longer flows into several short ones, so naturally smaller timeouts will yield a larger number of flows and a greater proportion of flows of smaller duration when analysing the traffic. However, even with a 2048 second timeout — which is essentially considered to be infinitive compared to the 3600 second data duration during which the data was captured — it was found in [6] that more than 27% of the flows consisted of a single packet of less than one hundred bytes. For timeout values of 64 seconds or less, 90% of the flows showed less than 50 packets, 5.5 kilobytes and 100 seconds of duration. From the data set it was known that the 80th percentile of the flows reflects about 40 packets or less and about 3.4 kilobytes of data or less. This led to the conclusion that a value of about 64 seconds should be reasonable to gather most of the flows.

The *NeTraMet* implementation, which we will introduce in chapter 3 uses a default value of 600 seconds for the timeout. This value can and should, depending on the traffic and host memory, be modified by the user.

It is evident that the choice of the flow timeout value always has to be dependent on the flow specification, the traffic that is to be measured and the memory that is available for the flow table. On a machine with lots of memory available that is used on a network segment where only a few flows per second would be measured, one would of course choose a much higher value for the flow timeout parameter than on a heavily loaded measurement point on a high–speed backbone network. In fact, for real measurement systems, it would eventually be interesting to do an automatic adjustment of this parameter so that always all of the available memory is used. However, no existing application has implemented such a mechanism yet.

2.5 Flows and Protocols

The flow methodology is not only interesting for measurement applications, it has also been discovered by protocol designers. As already mentioned in the introduction, flows can be used to accelerate switching and routing. To ameliorate the effectiveness that "*flow switching*" has, protocol designers have started developing special flow–oriented protocols. In the "IPng" (Internet Protocol Next Generation, IPv6) protocol for example a special field in the header has been reserved for future flow based applications. Another, more concrete development is Ipsilons flow management protocol, which is used to exchange flow–based routing policies between Ipsilon switches.

2.5.1 The IPng Flow Label

An important perspective for the use of flows comes from the the "IPv6" (also known as "IPng") protocol. It includes a 24–bit "flow label" field. This still remains somewhat experimental, but might in the future be used to simplify the determination which flow a packet belongs to.

The idea of this flow label field is that it may be set by applications to indicate the fact that some traffic is to be considered belonging to the same flow even if the application is exchanging the data without making use of the TCP (this would allow to identify the integrity of the flow by looking at TCP port numbers).

There are no rules how an application has to set the flow label field, it is completely left open to the programmer. Routers could one day make use of it, however since there are not yet any applications using it, it is not yet of much importance.

2.5.2 Ipsilon Tagged Flows

Ipsilons "*Flow Labelled IP*" approach to putting IP traffic on the ATM substrate [30] uses the concept of traffic flows to aggregate connectionless IP traffic onto a connection oriented ATM network.

What an Ipsilons flow–switch basically does is the following: It transparently monitors incoming IP packets and tries to detect flows of traffic matching the same criteria. Since it is already known that some kinds of traffic are only short–lived (for example DNS nameserver queries usually will not consist of more than some packets) the traffic can be categorized by TCP ports (i.e. by applications). For applications that generally will have longer duration traffic, flows are defined. For this, the fields in IP/TCP/UDP headers determining the routing decision, such as type of service, protocol, source address, destination address, source port, destination port etc. are used. Table 2.2 shows the classifcation that Ipsilon proposes for distinguishing the flows.

Flow–Oriented Traffic	**Short–Lived Traffic**
⋆ File transfer protcol (FTP) data ⋆ Telnet data ⋆ HyperText Transmission Protocol (HTTP) data ⋆ Web image downloads ⋆ Multimedia audio/video	⋆ Domain Name Service (DNS) queries ⋆ Simple Mail Transfer Protocol (SMTP) data ⋆ Network timing protocol (NTP) ⋆ Post Office Protocol (POP) ⋆ Simple Network Management Procotol (SNMP) queries

Table 2.2: Flow–oriented vs. short–lived types of internet traffic

Whenever a packet is received by the Ipsilon switch, it is reassembled and submitted to the control processor for forwarding. The processor forwards the packet in the normal manner, but it also performs a flow classifcation on the packet to determine whether future packets belonging to the same flow should be switched directly in the ATM hardware or continue to be forwarded hop–by–hop by the router software.

Flow classification depends on policies local to the switch. The flow classifier inspects the contents of the fields that characterize the flow and makes its classification decision based upon policies expressed in a table. Usually, the TCP port number will be used to identify the application that is responsible for the traffic. Thereby it is for example possible to configure the switch in a way so that flows belonging to FTP data connections will be switched but DNS queries will still be forwarded as datagrams.

The protocol that Ipsilon uses to exchange flow information between Flow Switches is called "*Ipsilon Flow Management Protocol*" (IFMP) and is specified in RFC 1953 [28]. The transmission of IPv4 datagrams over an ATM link is described in RFC1954 [27], both in a default manner or in the presence of flow labelling with IFMP.

In their paper [30], Newman et. al describe the performance of flow classification and they examine the suitability of standard internet applications for flow switching in detail. For this, they use the same data samples as Claffy, Braun and Polyzos did in [14], therefore the results should be comparable.

In the samples they examined about 84% of the packets and 91% of the bytes transferred were recognized as suitable for flow switching with the Ipsilon switch. They found that after an initial startup phase of 60 seconds about 92 flows per second were established. The average number of flows in the flow table was about 15,500. They also compared those results to a setup where all packets were classified for switching. Then a mean number of 422 flows per second would have to be established with an average of 42,000 entries in the flow table. Whether this could be done on an economic basis will surely depend on the architecture of the switch as well as on the price it costs to establish a new flow – however it leads to the conclusion that the effort to distinguish between different types of applications before the establishment of flows makes sense.

What is also worth mentioning is that Ipsilon switches already include a Web server and software that allows to configure and monitor the switch operation via a WWW browser. Because of this they are already well suited for the web–based enterprise management model that was presented in the introduction.

Chapter 3

Existing Flow–based Measurement and Analysis Applications

In this chapter, we will focus on existing applications for network measurement and analysis that are already using the flow concept. The concept is still quite new, so there are not yet too many of them. The most well known, since it is implemented in all newer Cisco switches is probably Ciscos "*NetFlow Data Export*". Also quite common is "*NeTraMet/NeMaC*", which is the first set of programs implementing the IETF architecture for traffic measurement. A more experimental project is NLANRs "OC3MON". This development is interesting because of its hardware which allows to analyze traffic on 150 Mbit/s ATM OC3 links carrying IP traffic which is encoded on top of ATM AAL5 as described in RFC1483 [10].

3.1 Cisco: NetFlow Data Export

Cisco "*NetFlow Switching*", which is available for the 75xx and RSP–7000 platforms in the IOS 10.3 and later releases includes an accounting mechanism that allows network managers to track network traffic on an end–to–end or per–application basis. The feature, which is referred to as "*NetFlow Data Export*", is using the same flow table which the switch already maintains for flow switching and exports it via a proprietary, connectionless protocol to a management PC or workstation. Flow descriptors are used to integrate route lookup, access filtering and IP accounting into one single fast lookup operation[1]. Figure 3.1 shows the architecture of the system. Since 100% of the traffic that is routed via the Cisco switch is assigned to flows in the flow table, it suffices for NetFlow Data Export to broadcast the information about each flow that is expired from the table to the management machine in order to account 100% of the transferred data.

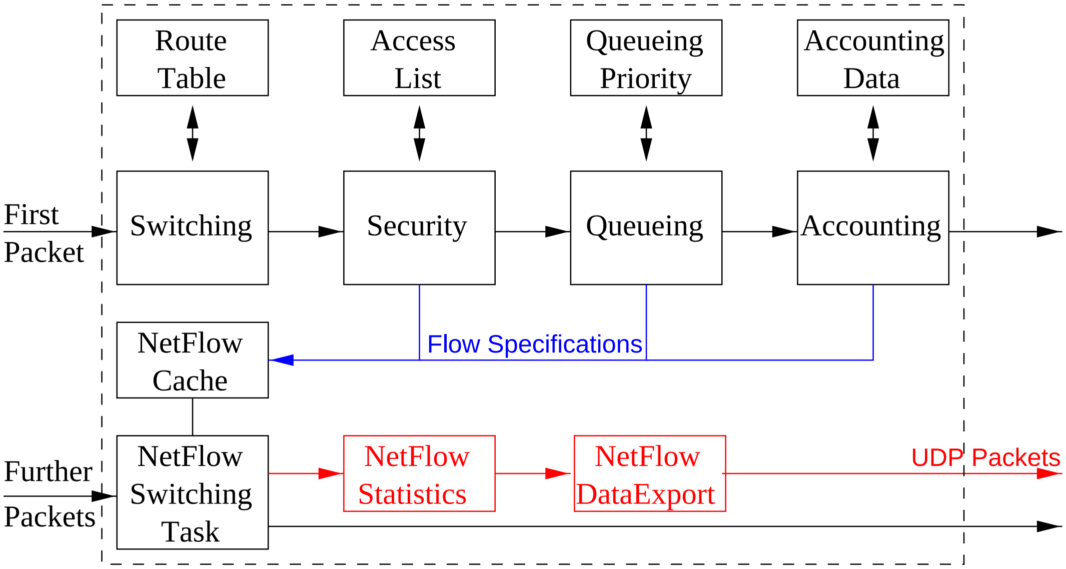


Figure 3.1: The Cisco FlowSwitching / FlowDataExport Architecture

However, the solution bears some problems:

- First of all, the connectionless (UDP) transmission of flow data can not guarantee that all data being broadcast is received by the managment station.
- The *flow specification* used by Cisco is *fixed* to IP addresses of source and destination machine plus port numbers. This makes sense for switching,

[1] Note that this only accelerates switching when access filtering and accounting are enabled on the switch

> however it may be a severe limitation when it comes to traffic monitoring and analysis. Far more data is produced and has to be transmitted via the connectionless path than it really would be needed for accounting.

A solution for the second problem might be to reduce the amount of data at a later stage by preprocessing it with custom programs, but on high speed ATM links the sheer amount itself can become a problem, especially since the management station usually is connected via a standard 10 Mbit/s ethernet or even only over a serial port. Because of this unability to use user–defined flow specifications, "*NetFlow Data Export*" won't probably scale very well for accounting, measurement and analysis applications.

It also has to be mentioned that Cisco has not deigned to say what criteria are used for flow identification and/or timeout. This makes it difficult to use the measurement data for scientific analysis. Nevertheless, the flow table maintained by the Cisco switches allows some interesting insights into Internet traffic characteristics. Figure 3.2 shows what kind of information can be measured. The measurements in this figure describe the traffic that is flowing through the BelWue core router in Stuttgart.

```
st1-gw#sh ip ca f
IP packet size distribution (13527M total packets):
   1-32   64   96  128  160  192  224  256  288  320  352  384  416  448  480
   .000 .314 .075 .055 .042 .014 .022 .012 .009 .011 .006 .045 .010 .004 .003

    512  544  576 1024 1536 2048 2560 3072 3584 4096 4608
   .003 .007 .171 .057 .128 .000 .000 .000 .000 .000 .000

IP Flow Switching Cache, 3356 active, 29412 inactive, 483496404 added
  180728533 tcp fin, 52221911 tcp rst, 157980484 timeout
  91934155 dns, 0 lru, 627965 counter wrap
  483493048 flows exported, 0 not exported, 21626175 export msgs sent
  flow alloc failures: 0 pkts, 0 bytes
  3 cur max hash, 1819 worst max hash, 3408 valid buckets
  0 tcp reordered flows, 0 reordered pkts, 0 syn retries
  0 tcp backed-off flows, 0 backoff pkts, 0 backoff secs
  statistics cleared 3091681 seconds ago

Protocol         Total  Flows   Packets Bytes  Packets Active(Sec) Idle(Sec)
--------         Flows   /Sec     /Flow  /Pkt     /Sec     /Flow     /Flow
TCP-Telnet     2085080    0.6        93    69     63.3      41.2      28.1
TCP-FTP        5132767    1.6        13    78     22.4      13.1      19.9
TCP-FTPD       4740235    1.5       246   472    378.4      21.1      12.0
TCP-WWW      205706062   66.5        14   349    982.0       7.0      10.3
TCP-SMTP       9677088    3.1        29   304     92.6      15.7      18.1
TCP-X           376666    0.1       264   280     32.1      35.6      31.1
TCP-BGP        1062162    0.3         3    49      1.0      11.4      32.3
TCP-Frag         22684    0.0        34   820      0.2      16.0      33.9
TCP-other     63125227   20.4        61   367   1260.8      35.9      16.4
UDP-DNS       91935276   29.7         3   105     97.6       0.9       6.9
UDP-NTP       14258676    4.6         2    75      9.5       0.9      34.0
UDP-TFTP          2522    0.0         4    52      0.0      12.3      34.0
UDP-Frag        176486    0.0        66  1360      3.7      44.9      33.8
UDP-other     60635750   19.6        10   157    203.3       6.6      33.9
ICMP          24030091    7.7         7    80     58.7       7.5      33.9
IGMP            450344    0.1       176   384     25.7     163.1      31.2
IPINIP           52300    0.0     33957   475    574.4     662.5      17.0
GRE              23598    0.0     94749   472    723.1    1464.2       1.7
IP-other           114    0.0        90    62      0.0       5.3      33.8
Total:       483493128  156.3        28   376   4529.5      10.3      15.8
```

Figure 3.2: NetFlow Statistics on the Stuttgart Core Internet Router

3.2 The IETF RTFM Working Group

The Internet Engineering Task Force (IETF) is a large open international community of network designers, operators, vendors, and researchers concerned with the evolution of internet architecture and the smooth operation of the Internet. It is open to any interested individual. The actual technical work of the IETF is done in its working groups, which are organized by topic into several areas (e.g., routing, transport, security, etc.). One of those working groups is the “Realtime Traffic Flow Measurement Charter”[2] (RTFM). This working group has three main objectives:

1. **Review existing work in traffic flow measurement**, including that of the RMON and internet accounting working groups and published work from independent researchers.
2. **Produce an improved Traffic Flow Model** considering at least the following:
   - Efficient hardware implementation
   - Effect of IPv6 on traffic measurement
   - Extension of the accounting model to widen the range of measurable quantities
   - Simpler ways to specify flows of interest
   - Maintain existing focus on data reduction capabilities
3. **Develop the Flow Meter MIB** as a “standards track” document with the IETF.

The groups current chairs are Nevil Brownlee (University of Auckland), Greg Ruth (GTE) and Sig Handelman (IBM). So far, the group has contributed several Internet Drafts [37]. Three of them have meanwhile become RFCs [4, 5, 26]. Those RFCs

[2] The RTFM charter provides a WWW homepage with further information on the current research state at `http://www.ietf.org/html.charters/rtfm-charter.html`

describe a proposal for an architecture for flow measurement (in [26]) as well as a formal specification for a traffic meter in ASN.1 notation, the “Meter MIB” [4]. Nevil Brownlee also provides a first implementation of the architecture in a set of programs called *“NeTraMet”* (Network Traffic Meter), *“NeMaC”* (NeTraMet Manager/Collector) and *“nifty”* (a graphical network traffic flow analyzer). In [5] he describes the experiences that were made with this set of programs.

3.2.1 The RTFM Architecture

The architecture is based on an earlier model described in RFC1272 [7]. It distinguishes between a traffic **meter**, a **meter reader** and a **manager**. Data is analyzed by an **analysis application** which is not further specified in the RFCs[3]. Figure 3.3 gives an overview over the architecture.

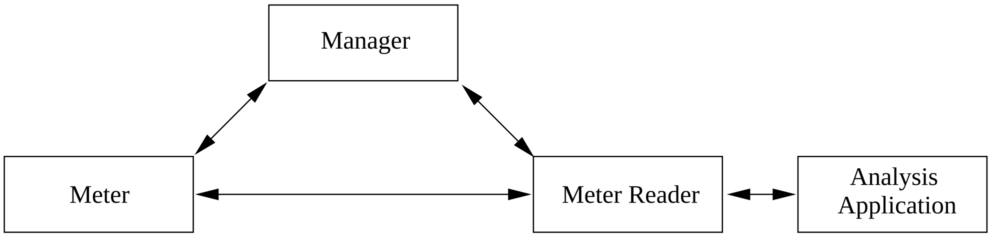


Figure 3.3: The RTFM Traffic Flow Measurement Architecture (RFC2063)

The following terms are defined by the RFC and describe the basic structure of the architecture:

Manager A traffic measurement manager is an application which configures “meter” entities and controls “meter reader” entities. It uses the data requirements of analysis applications to determine the appropriate configurations for each

[3] The architecture was initially designed for accounting purposes. The analysis application in an accounting scenario was the program that printed the bills for the customers.

meter and the proper operation of each meter reader. It may well be convenient to combine the functions of meter reader and manager within a single network entity.

Meter The meter is the heart of a traffic flow measurement system. Meters are placed at measurement points determined by network operations personnel. Each meter selectively records network activity as directed by its configuration settings. It can also aggregate, transform and further process the recorded activity before the data is stored. The processed and stored results are called the "usage data".

Meter Reader A meter reader reliably gets usage data from meters and makes it available to analysis applications.

Analysis Application An analysis application processes the usage data in order to provide information and reports which are useful for network engineering and management purposes. Examples include:

- **Traffic Flow Matrices**, showing the total flow rates for many of the possible paths within a network.
- **Flow Rate Frequency Distributions**, indicating how flow rates vary with time
- **Usage Data** showing the total traffic volumes sent and received by particular hosts.

3.2.2 IETF Example Applications: *NeTraMet*, *NeMaC* and *Nifty*

The *NeTraMet* Meter

NeTraMet, a meter for network traffic flows, is the first implementation of the architecture. It is available as public domain [4] software with source code and has been

[4] `ftp://ftp.auckland.ac.nz/pub/iawg/NeTraMet/`

successfully compiled on Solaris, SunOS, Irix and Linux. Under UNIX, the meter is using the `libpcap` BSD packet capturing library. A meter implementation on the PC (MS–DOS based) is also available. This implementation uses the CRYNWYR packet drivers for access to the networking hardware.

The roots of *NeTraMet* were not in traffic analysis, in the beginning it was developed for traffic accounting. Currently one Sun SPARC Workstation with *NeTraMet* is used for the traffic accounting of all IP traffic in New Zealands academic networks. The participiants in the New Zealand networks are charged fees based on the traffic measurements made with *NeTraMet*. The system has been in use for some years now and has shown to be a stable and suitable solution for this kind of measurement problem.

The meter could simply establish flows for every possible combination of source and destination attributes it observes (like the Cisco Switch does) but this would need a lot of memory. Instead the *NeTraMet* meter uses a set of **rules** to decide which flows are of interest. Those rules can also be set up in a way so that particular kinds of packets are totally ignored.

This is the main conceptual advantage that *NeTraMet* offers. It allows the use of *freely parametrizable flow specifications* for measurement, just as proposed by the model presented in chapter 2. For this purpose, so called "*rulesets*" can be defined that specify for which parameters the measurement is to be done. Those parameters could be adresses (of different layers), ports, or anything that one can imagine to categorize data from one flow against the others.

For the definition of traffic flow specifications, *NeTraMet* distinguishes the address attributes of a host by the following three kinds:

Adjacent On a standard ethernet this would be an ethernet MAC address. For other media, the decoding of addresses has not yet been implemented.

Peer A peer address can be an IP address, a DECnet phase IV address, a Novell network number, an EtherTalk address or a CLNS NSAP, these being the five protocols currently understood by *NeTraMet*.

Transport A transport address contains specifications for details within the peer protocol. For IP these are the protocol type and source and destination port numbers, and similar kinds of detail are defined for the other protocols.

This abstract definition of the flow endpoint granularities has the advantage that the RTFM architecture is easily extendable to new protocols. For example, the *NeTraMet* meter currently does not support the decoding of IPv6 or encapsulated IP. Nevertheless, implementing those protocols is possible within the architecture, so the only thing that has to be extended in order to support those protocols will be the applications.

Figure 3.4 shows how the abstract layers would correspond to transport, internetwork and network technology layer when the meter is used in a TCP/IP environment.

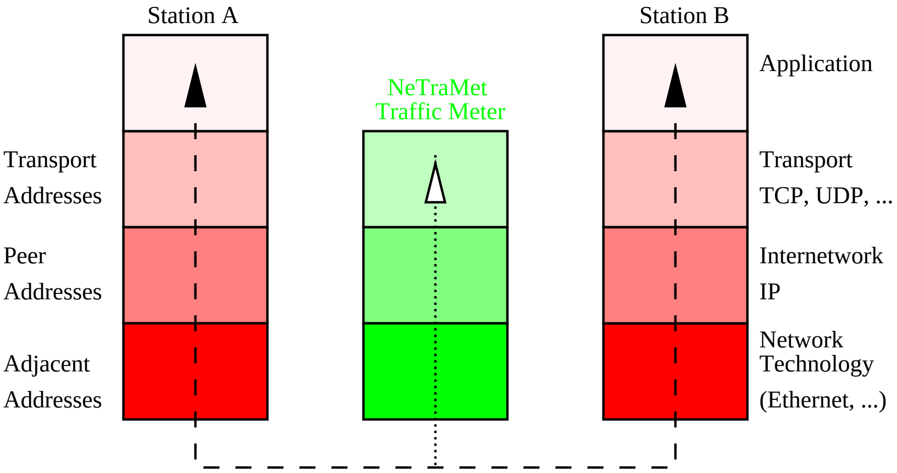


Figure 3.4: Different Layers *NeTraMet* can use for Flow Endpoints

Within the meter a *flow* is implemented as a data structure containing the attributes of its source and destination, its packet and byte counters, the times it was first and last observed, and other information used for control purposes.

This data structure is called a "`flowDataEntry`" node in the MIB. These entries are stored in a table under the OID

```
.mib-2.flowMIB.flowData.flowDataTable.
```

Table 3.1 shows the fields that are defined for each flow in those flow records. One of those records is allocated for each flow the meter captures. The records are stored in the "`flowDataTable`". The location of this table and the records in the `flowMIB` can be seen in figure B.1 in the Appendix.

In order to reduce the transfer time from the meter to the reader, the flow table itself can be transferred from the *NeTraMet* meter in a BER encoded packaged form with SNMPv2 GETBULK requests. Additionally, it is possible to read the entries in an unencoded form using standard SNMP GET requests.

Figure 3.5 shows how the times are defined for traffic flows captured by a *NeTraMet* meter. All times are measured relative to the uptime of the meter. Therefore, a meter reader will have to read the current uptime of the meter as well as all relevant time information for each of the flows that is to be analyzed. The two graphs depict the state of the same flow record, the second one showing the state a short time after the first. As it can be seen from the graph, additional packets have been received between the two queries.

For the "flow timeout" *NeTraMet* uses a default value of 600 seconds[5]. This value can be modified by meter managers by doing a SET operation on

```
flowMIB.flowControl.flowInactivityTimeout.
```

A flow that is not yet expired (i.e. for which the flow timeout has not yet been exceeded without the reception of a packet) has got the `flowDataStatus` "current". A flow for which no more data has been seen is "inactive".

When reading the flow data from the meter, one has to take care about the time skew that can occur during the query. For example if the meter reader would first read the meter uptime and then transfer the flow records for all flows it is interested in, it can happen that "`LastActiveTime`" values are seen that are in the future. When doing calculations with those times, this has to be kept in mind.

[5] This is the `flowInactivityTimeout` parameter in the MIB [4]

	Symbolic Name	**Type**	**Syntax**
1	flowDataIndex	Integer32	INTEGER (-2147483648..2147483647)
2	flowDataTimeMark	TimeFilter	INTEGER (0..4294967295)
3	flowDataStatus	INTEGER	
4	flowDataSourceInterface	Integer32	INTEGER (-2147483648..2147483647)
5	flowDataSourceAdjacentType	AdjacentType	INTEGER
6	flowDataSourceAdjacentAddress	AdjacentAddress	OCTET STRING [3..20]
7	flowDataSourceAdjacentMask	AdjacentAddress	OCTET STRING [3..20]
8	flowDataSourcePeerType	PeerType	INTEGER
9	flowDataSourcePeerAddress	PeerAddress	OCTET STRING [3..20]
10	flowDataSourcePeerMask	PeerAddress	OCTET STRING [3..20]
11	flowDataSourceTransType	TransportType	INTEGER (1..255)
12	flowDataSourceTransAddress	TransportAddress	OCTET STRING (2)
13	flowDataSourceTransMask	TransportAddress	OCTET STRING (2)
14	flowDataDestInterface	Integer32	INTEGER (-2147483648..2147483647)
15	flowDataDestAdjacentType	AdjacentType	INTEGER
16	flowDataDestAdjacentAddress	AdjacentAddress	OCTET STRING [3..20]
17	flowDataDestAdjacentMask	AdjacentAddress	OCTET STRING [3..20]
18	flowDataDestPeerType	PeerType	INTEGER
19	flowDataDestPeerAddress	PeerAddress	OCTET STRING [3..20]
20	flowDataDestPeerMask	PeerAddress	OCTET STRING [3..20]
21	flowDataDestTransType	TransportType	INTEGER (1..255)
22	flowDataDestTransAddress	TransportAddress	OCTET STRING (2)
23	flowDataDestTransMask	TransportAddress	OCTET STRING (2)
24	flowDataPDUScale	INTEGER	
25	flowDataOctetScale	INTEGER	
26	flowDataRuleSet	INTEGER	
27	flowDataToOctets	Counter64	INTEGER (0..18446744073709551615)
28	flowDataToPDUs	Counter64	INTEGER (0..18446744073709551615)
29	flowDataFromOctets	Counter64	INTEGER (0..18446744073709551615)
30	flowDataFromPDUs	Counter64	INTEGER (0..18446744073709551615)
31	flowDataFirstTime	TimeStamp	INTEGER (0..4294967295)
32	flowDataLastActiveTime	TimeStamp	INTEGER (0..4294967295)
33	flowDataSourceSubscriberID	OCTET STRING	
34	flowDataDestSubscriberID	OCTET STRING	
35	flowDataSessionID	OCTET STRING	
36	flowDataSourceClass	INTEGER	
37	flowDataDestClass	INTEGER	
38	flowDataClass	INTEGER	
39	flowDataSourceKind	INTEGER	
40	flowDataDestKind	INTEGER	
41	flowDataKind	INTEGER	

Table 3.1: Data Structure of Entries in the Flow Table of the *NeTraMet* Meter

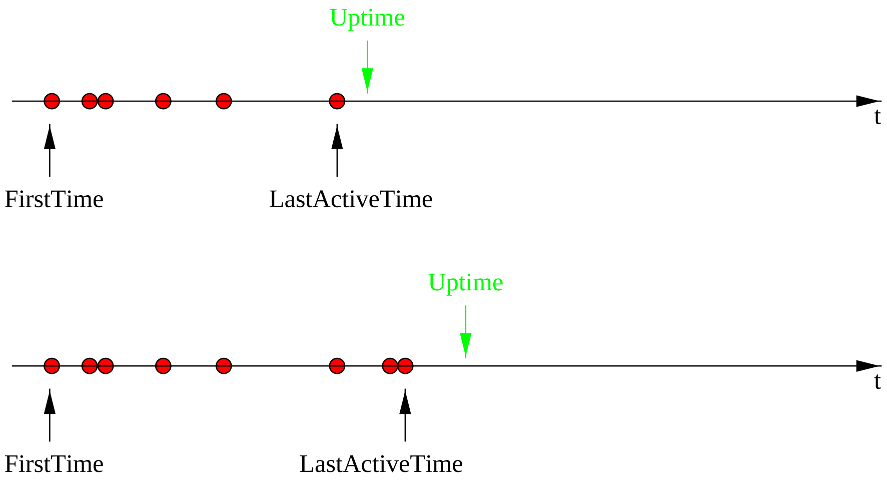


Figure 3.5: Flow Time Definitions with *NeTraMet*

The Manager/Collector “NeMaC”

NeMaC is a combined manager and collector for the *NeTraMet* meter. It is a simple Unix program, controlled via command line arguments. Parameters include the collection interval, rulefile name, configfile (determining which meters are to be controlled), SNMP community names and a number of optional operational parameters (like the flow timeout) for the *NeTraMet* meter which is to be managed.

While it is running, *NeMaC* generates a log file recording any unusual events observed for the meters being controlled as well as a “flows” file for each meter it controls.

The Traffic Flow Analyzer “nifty”

"nifty" is a Motif-based graphical interface that offers rapid insights into the current traffic on a network.

It displays a log–log plot which is updated after each sample period, i.e. each time the NeTraMet meter is read. Flow duration is plotted on the abscissa (=X axis). The

```
##NeTraMet v4.1:  -c120 -r rules.all  ksoc3mon oc3a  50000 flows
  starting at 12:56:58 Wed  2 Jul 1997
#Format: flowruleset flowindex firsttime  sourcepeertype sourcetranstype
 sourcepeeraddress sourcetransaddress to destpeeraddress desttransaddress
 topdus tooctets frompdus fromoctets
#Time: 12:56:58 Wed  2 Jul 1997 ksoc3mon Flows from 1 to 6625625
#Ruleset: 5  5 rules.all  NeMaC
#Stats: aps=8 apb=0 mps=24 mpb=0 lsp=0 avi=0.0 mni=0.0 fiu=6 frc=16
 gci=10 rpp=0.6 tpp=0.9 cpt=1.0 tts=49152 tsu=4
5 300 6625137  1 0
  141.46.14.72 1301 to 129.143.67.68 80
  1703936 203030528 1048576 457572352
5 301 6625140  1 0
  141.46.14.72 1304 to 129.143.67.68 80
  1441792 158859264 917504 343146496
5 304 6625182  1 0
  129.143.67.65 37939 to 194.195.240.82 80
  262144 49020928 262144 10485760
5 305 6625184  1 0
  129.143.67.65 37942 to 194.195.240.82 80
  393216 54788096 393216 398983168

  . . . .

#Time: 13:02:00 Wed  2 Jul 1997 ksoc3mon Flows from 6643806 to 6655878
#Stats: aps=26 apb=0 mps=134 mpb=0 lsp=0 avi=0.0 mni=0.0 fiu=246 frc=0
 gci=10 rpp=0.8 tpp=1.2 cpt=1.1 tts=57344 tsu=213
5 306 6625215  1 0
  204.70.74.61 4865 to 193.196.152.226 2299
  1835008 58720256 0 0
5 313 6625303  1 0
  129.69.47.58 4865 to 193.196.152.10 21997
  3932160 125829120 0 0
5 431 6639890  1 0
  128.182.73.68 3 to 193.196.152.226 3
  327680 18350080 0 0
5 434 6643418  1 0
  193.196.155.70 1600 to 128.182.73.68 4787
  262144 16777216 0 0

  . . . .
```

Figure 3.6: Sample flow data file as generated by NeMaC

duration d is calculated as

$$d = l - f$$

where l is the `LastActiveTime` and f is the `FirstTime` value from the record the flow meter keeps, as depicted in Figure 3.5. Each traffic flow is depicted as a symbol in the chart. The format can be specified within the rule file which nifty uploads to the meter upon startup. To display the different flow kinds, all ASCII characters as well as some symbols can be used. This is also configurable via the rule file.

Nifty plots

Via menu options it is possible to choose different display modes for the flow data. On the ordinate, nifty can display either packets or bytes. Additionally, the metric (rate, count or percent) can be choosen.

These choices give the user the following possibilities for analysing the network:

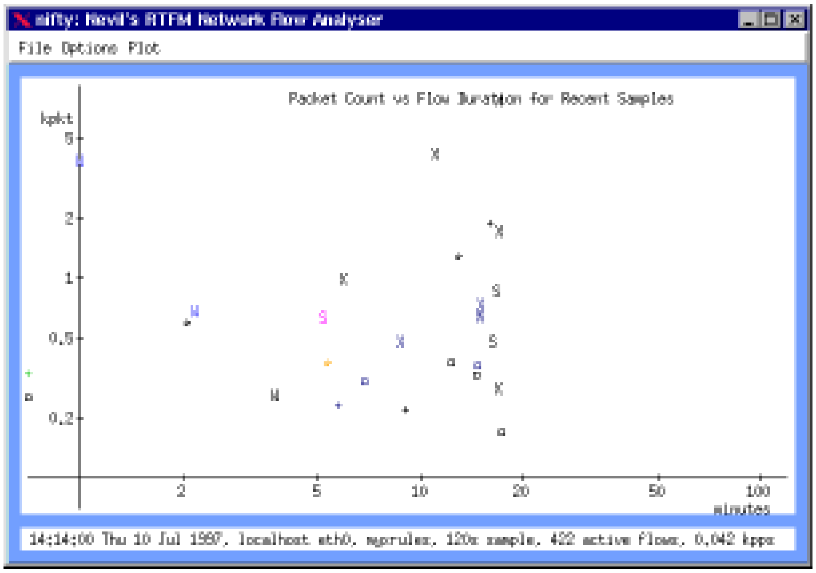
1. The **total number of packets** for each flow over the flow duration (i.e. the time between the first and the last packet seen for the particular flow) can be depicted, as shown in Figure 3.7.

2. In the same way, the "**packet rate**" r (i.e. the number of packets per second) can be depicted over the flow duration. The packet rate is computed as

   $$r = n/\tau$$

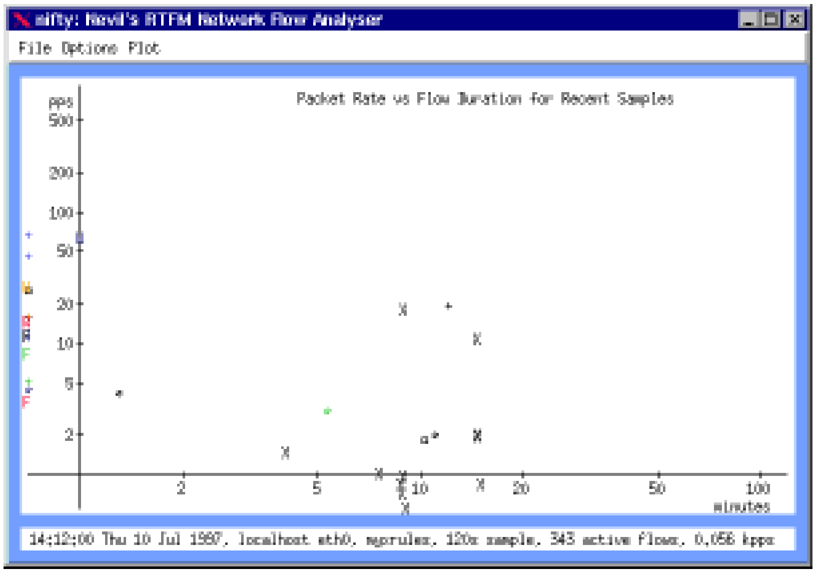
   where n is the number of packets observed in the last sample and τ is the length of the last sample. *This plot is most useful for observing flows of short bursts.* An example is shown in Figure 3.8.

3. Figure 3.9 shows the flow **packet rate as a percentage of the total observed packet rate** for the last sample.


nifty: Nevil's RTFM Network Flow Analyser
File Options Plot
Packet Count vs Flow Duration for Recent Samples
kpkt
5
2
1
0.5
0.2
2
5
10
20
50
100
minutes

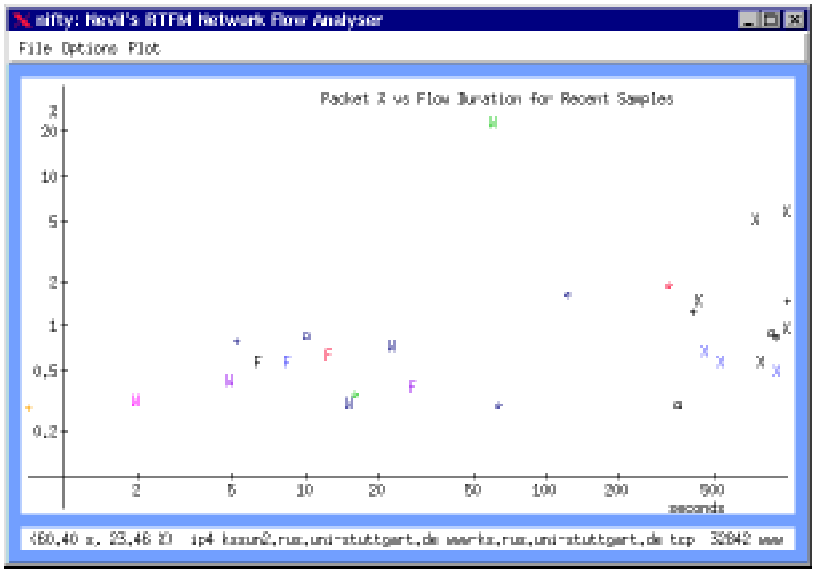
14:14:00 Thu 10 Jul 1997, localhost eth0, mgrules, 120s sample, 422 active flows, 0.042 kpps

Figure 3.7: Nifty displaying the packet count vs. the flow duration


nifty: Nevil's RTFM Network Flow Analyser
File Options Plot
Packet Rate vs Flow Duration for Recent Samples
pps
500
200
100
50
20
10
5
2
2
5
10
20
50
100
minutes

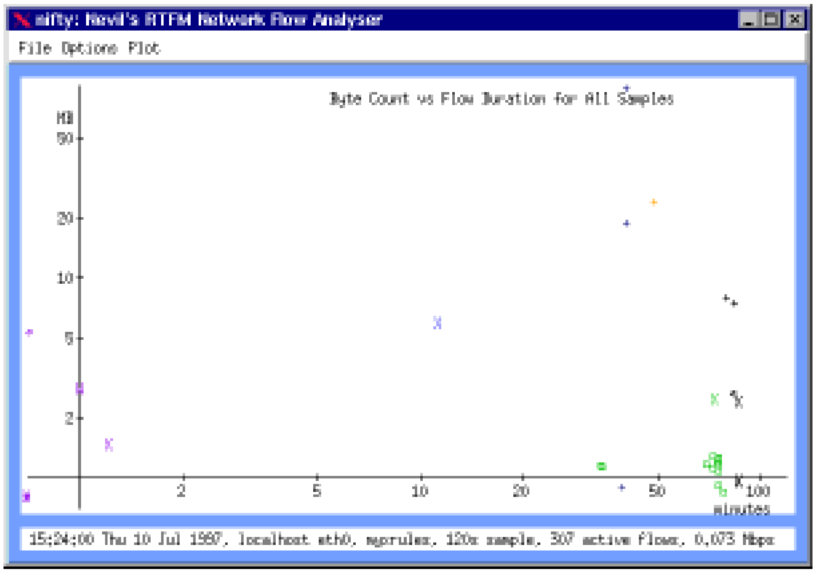
14:12:00 Thu 10 Jul 1997, localhost eth0, mgrules, 120s sample, 343 active flows, 0.056 kpps

Figure 3.8: Nifty displaying the packet rate vs. the flow duration


nifty: Nevil's RTFM Network Flow Analyser
File Options Plot
Packet % vs Flow Duration for Recent Samples
%
20
10
5
2
1
0.5
0.2
2
5
10
20
50
100
200
500
seconds

(60.40 s, 23.46 %) ip4 kssun2.rus.uni-stuttgart.de wwwrks.rus.uni-stuttgart.de tcp 32842 www

Figure 3.9: Nifty displaying the packet percentage vs. the flow duration


nifty: Nevil's RTFM Network Flow Analyser
File Options Plot
Byte Count vs Flow Duration for All Samples
MB
50
20
10
5
2
2
5
10
20
50
100
minutes

15:24:00 Thu 10 Jul 1997, localhost eth0, myrules, 120s sample, 307 active flows, 0.073 Mbps

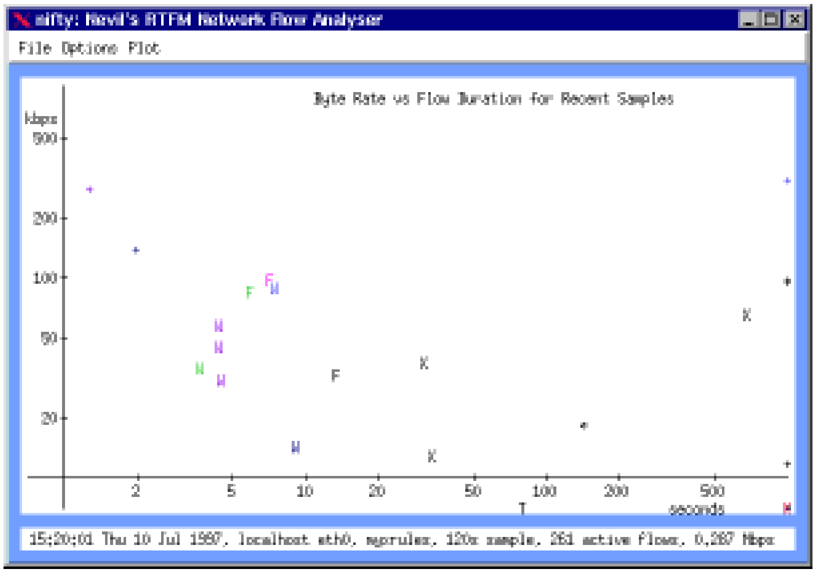
Figure 3.10: Nifty displaying the byte count vs. the flow duration

4. The **total number of bytes** can be displayed over the flow duration, as in Figure 3.10. *This plot is most useful for observing long–term high–volume flows.*

5. Figure 3.11 shows how the number of **bytes per second**, estimated from the counts and times observed for the last sample, can be displayed over the flow duration.

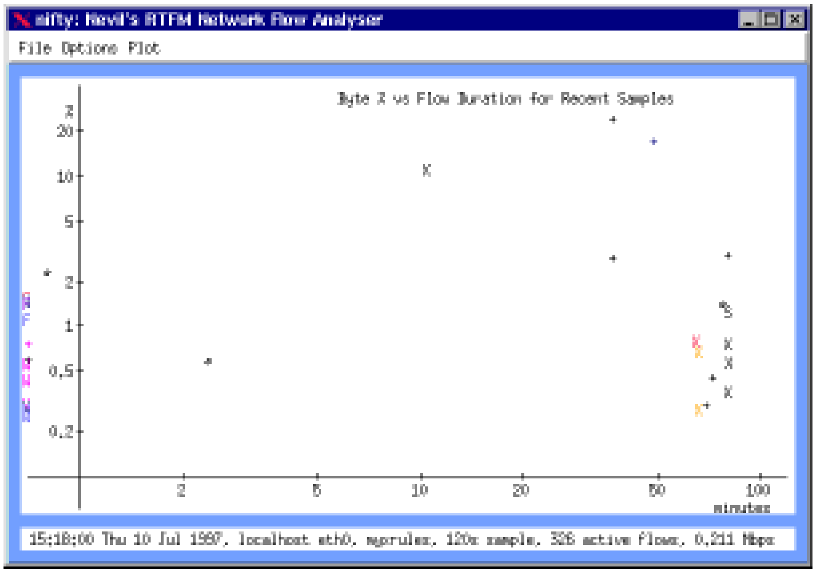
6. The **flow byte count** can be displayed as a **percentage of total observed bytes** for the last sample. Figure 3.12 shows an example for this.

These methods to display the traffic information have shown to be very valuable for getting rapid insights into the current state of the network. Once one gets used to them, they allow the user an immediate perception of the current traffic. Traffic sources producing excess packets can be easily located in the chart. At the same time, the different characters representing protocols allow a instantaneous overview of the kinds of applications that are responsible for the network load.


nifty: Nevil's RTFM Network Flow Analyser
File Options Plot
Byte Rate vs Flow Duration for Recent Samples
kbps
500
200
100
50
20
2
5
10
20
50
100
200
500
seconds

15:20:01 Thu 10 Jul 1997, localhost eth0, myrules, 120s sample, 261 active flows, 0.287 Mbps

Figure 3.11: Nifty displaying the byte rate vs. the flow duration


nifty: Nevil's RTFM Network Flow Analyser
File Options Plot
Byte % vs Flow Duration for Recent Samples
%
20
10
5
2
1
0.5
0.2
2
5
10
20
50
100
minutes

15:18:00 Thu 10 Jul 1997, localhost eth0, myrules, 120s sample, 326 active flows, 0.211 Mbps

Figure 3.12: Nifty displaying the byte percentage vs. the flow duration

3.3 The NLANR OC3MON

In order to measure traffic at higher speeds like ATM OC3 (150 Mbits/s), data capturing can no longer be effectively done on standard UNIX machines using mechanisms as the `libpcap` library. Therefore, the National Laboratory for Applied Network Research (NLANR) and MCI Telecommunications Corp. have developed OC3MON as a low cost high–speed traffic flow measurement system [13]. OC3MON is part of the "CORAL" project, which is now aiming at the development of a similar monitor for even higher (OC12) line speeds.

3.3.1 OC3MON System Overview

The system, whose hardware is depicted in Figure 3.13 consists of an IBM personal computer clone with 128 MB of main memory, a 166 MHZ Intel Pentium processor, an Ethernet interface and two Fore PCA–200 PCI ATM interface cards running on a 33 MHz 32–bit wide PCI bus. The Intel i960 processor on the Fore cards allows to optimize OC3MON operation with custom firmware. Usually, Fore does not disclose the source code for this firmware, however NLANR made arrangements with Fore to obtain it and freely distribute the custom firmware executables along with the source code developed for the OC3MON system processor.

The OC3MON ATM NICs are attached to an OC3 fiber pair carrying IP traffic, connecting the receive port of each ATM card to the monitor port of an optical splitter, which carries 5% of the light from each fiber to the receive port of one NIC. Attached to an OC3 trunk terminated on a switching device (e.g., ATM switch or router), one of the OC3MON NICs sees all traffic received by the switching device and the other NIC sees all traffic transmitted by the switching device.

The system is currently used on the vBNS in between the wide area ATM backbone and the primary nodes at the supercomputer centers [6]. Other sites using OC3MON

[6] vBNS/NLANR provide the measurement data as well as information on the state of the CORAL project to the public at `http://www.nlanr.net/NA/Oc3mon/`

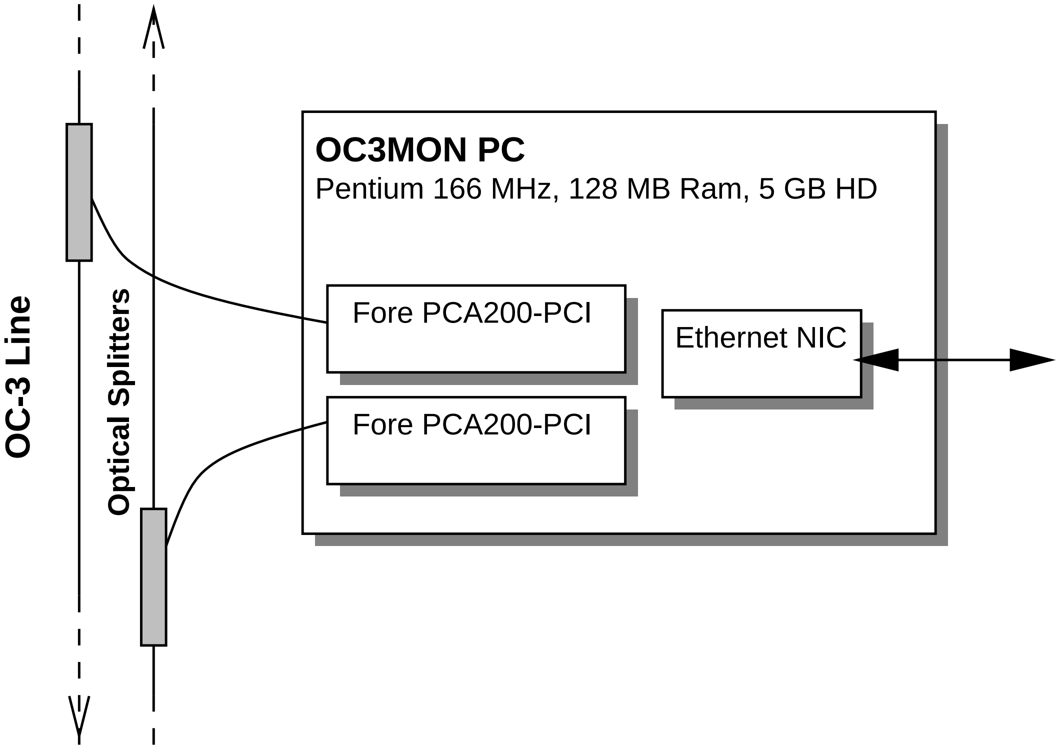


Figure 3.13: The OC3MON Hardware

include the National Center for Supercomputing Applications (NCSA) at the University of Illinois, Urbana–Champaign, Nokia (Finland), NTT (Japan), the University of Auckland (New Zealand) and MCI (USA).

The software for OC3MON is free. All source code files are available by FTP[7]. However the source of OC3MON is relatively difficult to reuse or extend, since it strongly depends on the Fore adapters that are used, for which the programmers manuals are not available without special agreements with Fore.

[7] The source code is available at `ftp://nlanr.net/Software/Oc3mon`

3.3.2 The OC3MON Software

The DOS–based software running on the PC consists of device drivers and a TCP/IP stack combined into a single executable; higher level software performs the real-time flow analysis. DOS was chosen as operating system, because Unix has a higher interrupt latency. Since the Texas Instruments cards in the original OC3MON design required polling at 1/128 the cell rate in order to obtain accurate timestamp granularity at full OC3 rate (the card itself did not timestamp the cell), a very fast response to interrupts was necessary. Even if the two cards would have been put into two very fast Pentium PCs, timing would still have been critical on Unix Systems which cannot guarantee realtime response.

The latest OC3MON design uses Fore cards that can attach timestamps to the cells on their own; the host no longer needs to poll the card at all. Interrupts only occur at most every 1/40 second (e.g., if both links received 40 byte packets simultaneously), so low latency is no longer a constraint. Therefore now there is a first effort to port the OC3MON software to a UNIX environment[8].

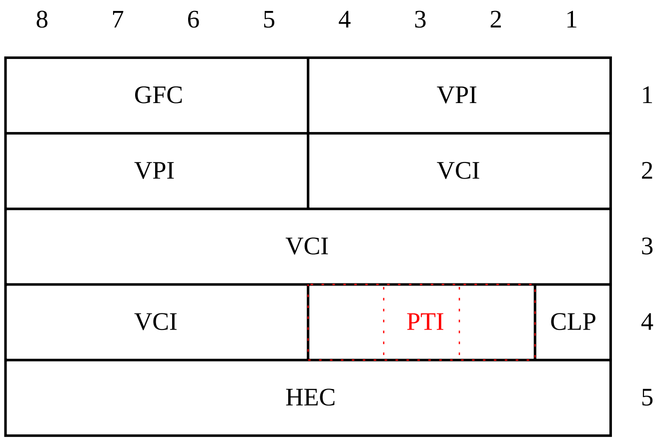
Background on ATM

Several possibilities exist to route IP traffic over ATM networks [33]. The OC3MON software was designed to work only with IP traffic encoded using LLC/SNAP encapsulation (cf. [3], p. 208) conforming to RFC1483 [10] (i.e. on top of ATM AAL5 [9]).

[8] Jon Dugan, `jdugan@ncsa.uiuc.edu` is currently working on an OC3MON port to FreeBSD. He maintains a web page at `http://rivendell.ncsa.uiuc.edu/oc3mon`

[9] Note that Cisco routers and Fore switches also support AAL3/4, but this is not used very often because it consumes an additional 4 bytes from each cell (above the 5 already used for the ATM header) to support submultiplexed channels within a given VP/VC. Therefore it was not implemented in OC3MON
Since the LLC/SNAP 8–byte per–frame header that the routers insert already includes a 2–byte ethertype field that allows, if needed, multiplexing of different protocols (IP, IBM SNA, Novell IPX, etc) on the same VC, including AAL3/4 support in the design would not have been beneficial. Note

The software directs each ATM card to perform AAL5 reassembly on a specified range of virtual circuit and virtual path identifiers (VCI/VPI).



GFC:	Generic Flow Control	PTI:	Payload Type Identifier
VPI:	Virtual Path Identifier	CLP:	Cell Loss Priority
VCI:	Virtual Channel Identifier	HEC:	Header Error Control

Figure 3.14: ATM Header Structure at the UNI

AAL5 Frame Reassembly using VPI, VCI and the PTI fields of the ATM Header

AAL5 makes use of the "*end of SAR–SDU*" indication, which is carried as a value 1 in the lowest bit of the ATM layer user to ATM layer user indication bit field of the PTI field in the ATM header to indicate whether a cell is the last in a frame[10]. Figure 3.14 shows where the PTI field is located inside the ATM UNI header structure. The payload data is encapsulated in the 48 bytes that follow this header.

AAL5 also *assumes that cells for a given frame will not be interspersed with cells for another frame within the same VP/VC pair.* Combined with a single bit of state per VP/VC pair maintained by the receiver, which indicates that the cell is in the

also that the addition of any bytes to a simple (no data attached) TCP ACK causes the 8-byte ATM AAL5 trailer to require another cell, doubling the bandwidth used by such packets.

[10] cf. [8], p. 138

middle of a frame for that VP/VC pair, there is enough information to reassemble the frame[11].

Since OC3MON has no need for data beyond the first cell, and since it already maintains per–flow state on the host, the per–VC state on the card is limitted to the bare minimum: one bit.

This limit allows to use 20 bits for VPI and VCI information, yielding a 128KB table size. Although the Fore cards have 256KB of memory, some of it is used for the i960 code (about 32K), the OS, reassembly engine data structures, and the stack. Since the VP/VC lookup needs an exact power of two, the largest possible number of VP/VC pairs is 2^{20}. (Single-bit state for 2^{20} VP/VC combos = 2^{17} bytes = 131072 bytes = 128KB).

Examining twenty bits of VCI/VPI information allows OC3MON to monitor over one million VCs simultaneously. The host controls exactly how many bits of the VCI this 20–bit index will include; the rest derive from the VPI. The host also specifies at startup what to expect for the remaining bits of the VPI/VCI, i.e., those not used for indexing into the card's state table. The card can then complain about, or at least drop, non-conforming cells.

Since OC3MON is designed to be able to see traffic on (almost) any VPI/VCI without prior knowledge of which circuits are active and because the fast SRAM (static random access memory) used on such ATM cards for state tables is expensive and not amenable to modification by the consumer, this design turned out to be extremely advantageous.

Software–Description

The AAL reassembly logic is customized to capture **only the first cell of each frame** as shown in Figure 3.15. The 48 bytes of payload from these cells typically

[11] Note that when multiple cells of the same packet are copied to the host, the card will not place them near each other so the host must do further reassembly using the ATM headers.

contain the LLC/SNAP (8 bytes), IP and TCP (typically 20 bytes each) headers[12]. Copying the 5 byte ATM header also allows the flexibility of doing ATM based analysis in the future. The SAR engine discards the rest of each AAL5 protocol data unit (PDU, equivalent to a frame or IP packet), limiting the amount of data transferred from the NICs over the PCI bus to the host. Although as yet unimplemented, one could increase the amount collected to accommodate IP options or larger packet headers as specified for IPng. Currently, however, the cards only pass the first cell of each packet, so when IP layer options push part of the TCP header into the second cell, these latter portions will not be seen by the host.

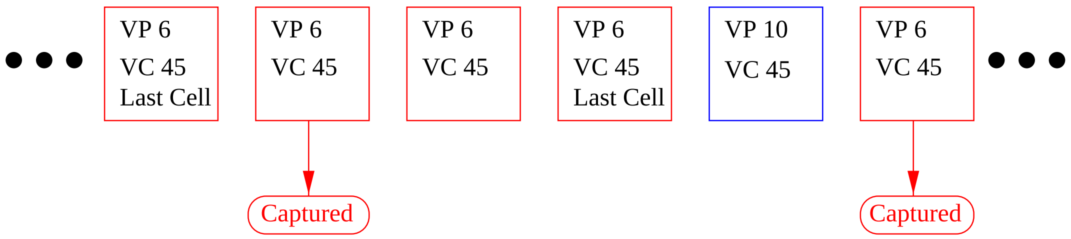


Figure 3.15: Which ATM Cells are Captured

Each ATM card has two 1MB buffers in host memory to hold IP header data. The cards are bus masters, able to DMA (direct memory access) header data from each AAL5 PDU into the host memory buffers with its own PCI bus transfer. This capability eliminates the need for host CPU intervention except when a buffer fills, at which point the card generates an interrupt to the host, signaling it to process that buffer up to memory while the card fills the other buffer with more header data. This design allows the host to have a long interrupt latency without risking loss of monitored data. The cards add timestamps to the header data as they prepare to transfer it to host memory. Clock granularity is 40 nanoseconds, about $1/70$ of the OC3 cell transmission time.

[12] Note that encapsulated IP traffic cannot be analyzed since the encapsulated headers do not fit into the first cell

Usage

For data analysis, the OC3MON can be used in two different modes:

1. Two **raw capture modes**[13], in which all packet headers are captured to memory and then dumped to disk.

   (a) The first of these modes captures the complete first ATM cell in each packet, which includes TCP/IP headers.

   (b) The second one captures only the ATM headers, but not only for the first cell of each packet but for all cells.

   Raw capturing is useful for getting a detailed view of traffic over a relatively brief interval. This allows an extensive further analysis of the captured data. However, because currently the DOS–supplied disk I/O routines — which are blocking — are used, it is not possible to write to disk simultaneously with data capturing. In fact, the I/O is not even fast enough to sustain disk transfer of a packet trace without the flow analysis process running. Therefore one can only collect a trace as big as the size of host memory, which in our case would be 114MB (119.5 million bytes), and then must stop OC3MON header collection to let OC3MON transfer the memory buffer to disk. In the future, separate I/O routines that directly use the hardware will eventually be developped, bypassing the slower DOS routines and allowing to keep up with OC3 line rate.

2. An **IP flow capture mode** in which the OC3MON maintains flow statistics that do not require the storage of each header.
   Because the amount of data captured in a packet level trace and the time needed for our disk I/O inhibits continuous operational header capture, this mode of operation is the default mode. Once again this shows the advantages

[13] Apparently, a third mode has been added in July 1997 to capture all of all cells. The author had no possibility to test this yet.

> that the concept of flows offers for traffic analysis. Concurrently with the interrupt driven header capture, software runs on the host CPU to analyze the packet headers and to establish flows. The flows are analyzed and stored at regular intervals for remote querying. For querying, a Perl script on a web server is executed in regular intervals and the data is captured to a file. The queries themselves are done with simple `telnet` type connections to port 22 of the OC3MON PC. Using port 22 has the advantage that this port often is not blocked by firewalls.

The way how queries in the flow capture mode are implemented is the main disadvantage of the OC3MON software. Using `telnet` on the one hand is simple to implement, on the other hand the data is transferred in a non–standardized ASCII format. Every change to the OC3MON software requires a change in the applications that use OC3MON as well. This makes it difficult for application developers to integrate OC3MON in an own environment.

Additionally by using `telnet`, security is not present in the OC3MON querying architecture at all. Everyone with IP access to the machine can query it with a simple `telnet` command. What is even worse, those queries will reset the counters on OC3MON and therewith the automated measurements will display wrong data afterwards. These security problems can currently only be solved by putting the OC3MON ethernet interface inside a secure network. For those reasons, the querying protocol is a severe limitation for practical use of the software. However, we will later see (in section 3.4) that there is already a promising solution for this.

Is OC3MON really fast enough for 150 Mbits/s?

OC3MON was tested by NLANR on an OC3c link fully occupied with single cell packets (as would occur in the admittedly unlikely event of continuous TCP ACKs with no data and LLC/SNAP disabled on the routers), which yields 353207.5 packets per second (or in the single-cell packet case, the same number of cells) across

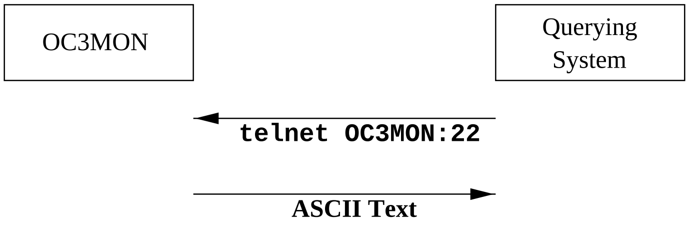


Figure 3.16: How Data is retrieved from OC3MON

each half-duplex link. Each header, including timestamp, ATM, LLC/SNAP, IP and TCP headers consumes 60 bytes, so the internal bus bandwidth required would be

$$353207.5 * 2 * 60 * 8 = 339 \text{ Mbit/s.}$$

The 32-bit, 33 MHz bus in the PC is slated at 1.056 gigabits, so bus bandwidth will not be a bottleneck until OC12 is to be supported.

Generic internet environments exhibit average packet sizes closer to 250 bytes (about 5 cells), and rarely full utilization in both directions of a link. If we estimate 66% utilization in one direction and full utilization in the other, we get a more realistic:

$$353207.5 * 1.6666/5 = 117731$$

headers per second, or 56.5 Mbits per second across the internal bus. This estimation shows that OC3MON can be used to monitor a full–duplex 150 Mbit/s ATM connection without suffering any capacity problems.

3.3.3 Flow Definition the OC3MON uses

The OC3MON works with a flow model as it was described by Claffy, Braun and Polyzos in [14]. As proposed in the paper, the OC3MONs flow timeout value defaults to 64 seconds. This value can however be changed using a command line parameter upon program startup. Afterwards, the value cannot be changed without restarting the whole program.

3.3.4 Flow Specification Criteria with the OC3MON

The granularity for flow endpoints has to be chosen upon start of the OC3MON software on the PC. It cannot be changed at runtime. The following *flow criteria* are possible:

“`sh`’’:	Source Host Only
“`dh`’’:	Destination Host Only
“`hp`’’:	Host Pair
“`sn`’’:	Source Network Only
“`dn`’’:	Destination Network Only
“`np`’’:	Net Pair
“`pq`’’:	Address/Port quadruples

Additionally, an analysis that uses autonomous system (AS) numbers instead of IP addresses is possible with the latest versions of OC3MON. This allows network operators to gain interesting insights in who (which countries etc.) is actually using their infrastructure.

3.4 OC3MON with NeTraMet Statistics Module

Until december 1996 OC3MON has been a stand–alone development. Only recently, Nevil Brownlee has written a *NeTraMet* statistics module[14] for OC3MON which is now part of the OC3MON distribution.

The "OC3MON–NeTraMet" has interesting advantages compared to the "traditional" OC3MON software:

- The interface is standardized. This makes it easier to develop own applications for OC3MON.
- Using SNMP also allows to integrate OC3MON into existing network management environments.
- By using SNMP, now at least a simple security mechanism is implemented.
- The sophisticated flow–measurement tools (*nifty*/*NeMaC*) that have been developed for use with *NeTraMet* are now usable on OC3 lines as well.
- Existing accounting solutions that have been implemented for ethernet can now be used on ATM OC3 without any modifications.
- The *NeTraMet* architecture with its "meter manager" entities gives the meter reader much more control over granularity, since it allows the upload of user–defined rulesets to the meter. On the traditional OC3MON it was only possible to set the granularity with a command line switch, and not too many possibilities were implemented.

A disadvantage of the *NeTraMet* mode for OC3 certainly is that it is harder to write a web interface using SNMP than to write one using the `telnet` command to query the monitor. Nevertheless it is surely worth the additional effort, since those applications written for the *NeTraMet* environment will be of use for a much larger

[14] Nevil Brownlee describes statistics modules at `http://www.nlanr.net/NA/Oc3mon/oc3-api.html`

community. Additionally, *NeTraMet* and its structure have matured for a long time which assures that the mechanisms and protocols are working reliably.

3.5 Feature Overview

The following table gives an overview over the features beeing offered by the solutions presented in the last sections. Since OC3MON with *NeTraMet* Statistics Module basically is an OC3MON with *NeTraMet* as interface, it does of course offer more or less the same features as the standard *NeTraMet* does. However there are some restrictions due to the fact that the RFC1483 ATM OC3 link is not usable for the same kinds of encapsulations as a standard ethernet would be. Therefore, this solution is presented in a separate column of Table 3.2.

	NetFlow DataExport	**OC3MON**	**NeTraMet**	**OC3MON & NeTraMet Statistics Module**
Organization / Manufacturer	Cisco	NLANR/MCI	IETF	NLANR, IETF
Possible Flow Specifications				
Host Pair + Port Pair	y	y	y	y
Host Pair only	n	y	y	y
Source Host only	n	y	y	y
Destination Host only	n	y	y	y
Network Pair	n	y	y	y
Source Network Only	n	y	y	y
Destination Network Only	n	y	y	y
Autonomous Systems	n	y	n	n
Arbitrary Groups	n	n	y	y
Flow Timeout				
Default Value	not specified	600s	64s	64s
Configurable	n	y	y	y
Network Technologies				
Ethernet	n	n	y	n
ATM–OC3 RFC1483	n	y	n	y
FDDI	n	n	y	n
Known Protocols				
IPv4	y	y	y	y
IPv6	n	n	n	n
Encapsulated IP	n	n	n	n
TCP	y	y	y	y
Novell IPX	n	n	y	-
EtherTalk	n	n	y	-
DECnet	n	n	y	-
ISO CLNS	n	n	y	-
Security Mechanism	should be put behind a firewall	UDP broadcast only to specific address	SNMPv2 authentication	SNMPv2 authentication
Querying Protocol	Cisco Proprietary	ASCII / telnet	SNMP	SNMP

Table 3.2: Overview of the presented Flow–based Applications

Chapter 4

Writing Web–based Management Programs

In this chapter, the necessary techniques for writing web–based network management programs are presented. First, a brief introduction into the “Simple Network Management Protocol” (SNMP) is given. Then different methodologies that can be used for web–based programming (CGI, Java) are compared. Since for our needs Java is the best solution, Java is further introduced. The focus in this introduction is the AdventNet SNMP class library that can be used to write SNMP applications in Java.

4.1 The Simple Network Management Protcol (SNMP)

The “Simple Network Management Protocol” (SNMP) [36, 38] is a protocol for internet network management services. It is formally specified in the IETF standards STD15 [23], STD16 [15], STD17 [16] and STD50 [17] as well as a series of RFC documents. The protocol is not limited to use on TCP/IP networks. It can for example also be used over OSI [35] or on IPX [2].

4.1.1 Architecture

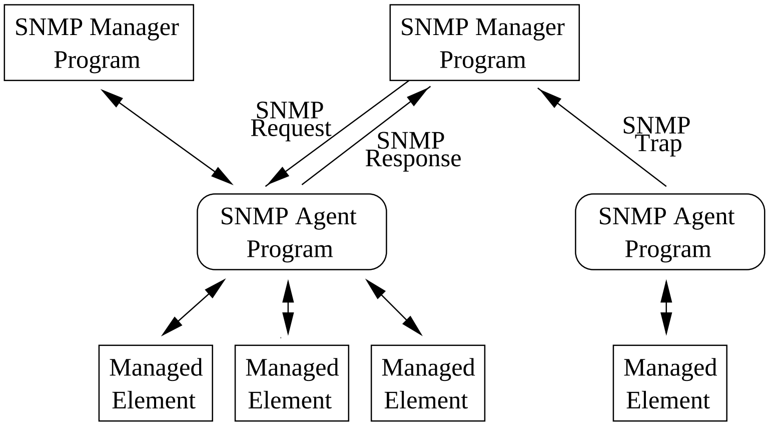


Figure 4.1: SNMP Agents and Managers

Figure 4.1 depicts the basic architecture. Management is based on a client–server model. The "*manager*" is a client process which communicates with managed nodes. These nodes provide information via server processes called "*agents*". RFC 1470 [32] contains an overview over existing SNMP manager products for different platforms.

Agents can be contacted by multiple managers. The manager contains the management functions and presents management information to the user. It perceives the agent as a virtual data store that is populated with instances of “managed objects” (MOs), as shown in Figure 4.2. A managed object is a parameter or an attribute that the agent monitors. Instances of the managed objects are simply the current values of the parameters or attributes.

A manager program sends requests to one or more agents. The agent returns the requested information in response. Each request causes the agent to generate exactly one corresponding response.

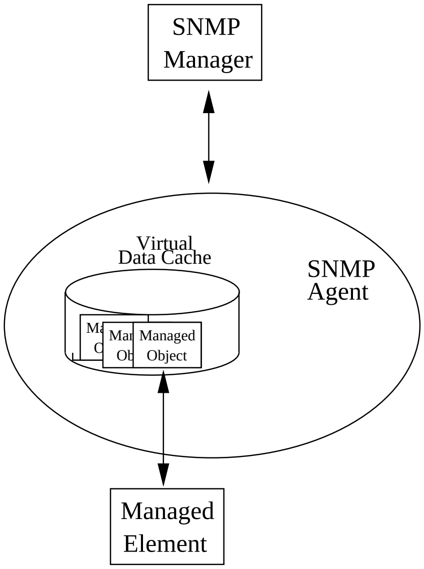


Figure 4.2: SNMP Agent Functionality

4.1.2 Alarm Messages (Traps)

The SNMP agents can also initiate communication with the managers by sending so called “*Traps*” [34]. This can for example be used to notify the network operator upon special events like failures of links etc.

A connectionless transport protocol, UDP, is employed to convey SNMP messages between the agents and managers. Even though with this protocol delivery is not guaranteed, the reality is that a vast majority of the messages are successfully delivered. In addition most manager programs keep track of requests that they have sent. If a response is not received within a time–out interval, the manager program retransmits the request. However, using an unreliable protocol to carry SNMP has one disadvantage when transmitting the unsolicited trap messages from an agent to a manager. The agent has no way of determining if a trap was actually received by the intended manager (the manager may not even be running!).

4.1.3 SNMP Proxy Agents for the Management of Non–SNMP Devices

In Figure 4.1, the SNMP agent is assumed to be part of the managed hardware. In the real world there are resources that either are not managed by a computer process or are managed using a non–SNMP protocol. Still many device manufacturers either do not provide support for SNMP or they may provide limited SNMP support. The latter includes switch vendors that have SNMP agents on some of their interface cards, but do not have agents that monitor the entire switch, including power supplies, the CPU, and the configuration. In this case if you want to use SNMP to manage the switch, you might need a so called "*proxy agent*" to monitor and control the switch. Such proxy agents allow access to remote information about passive components (cables, transceivers, repeaters) and resources of other protocol environments.

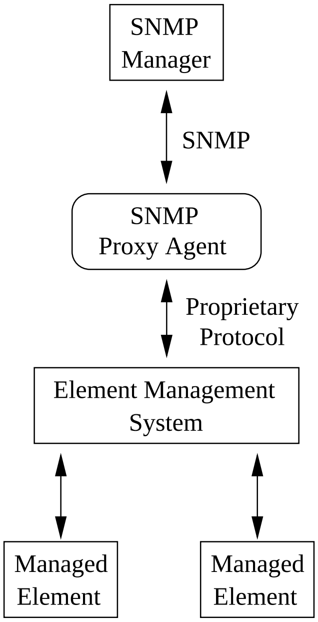


Figure 4.3: SNMP Proxy Agent

Figure 4.3 depicts an example for such a scenario. The protocol between the SNMP proxy and the elements management system depends on the specifications of the latter, and usually will be a proprietary one.

4.2 Programming for the WWW

To write programs that use a WWW browser as their user interface, two main possibilities exist:

CGI ("Common Gateway Interface") programs can be written in any programming language that is available on the web–server. The CGI program can be called upon specific events by the web–server. Often, such programs are simple shell or Perl scripts. The CGI provides a common interface between the server and the program by passing arguments inside environment variables. The program in return has to create HTML code as output which will be sent to the web browser of the user who originated the event that caused the CGI program to start.

Since standard HTML code is generated, programs that use the CGI technique can be executed using all kinds of web–browsers. It suffices that the browser is capable of displaying the generated HTML code.

Java programs are, in contrary to CGI programs, executed inside the Java Virtual Machine (VM) [24] of the users web browser. To make this possible, Java source code is compiled into so called "class files" in a special Java byte code format. After beeing transferred to the web browser on the client machine, the Java byte code is verified for security problems and then executed within the Java Virtual Machine. The same byte code format is used on all implementations of the Java VM, therefore an application written in Java is executable on all kinds of platforms for which a Java VM is available. The VM is included in all recent versions of the Netscape Web browser, therefore Java programs can be executed by virtually everybody who uses the World–Wide–Web.

From the users view, Java programs when compared to CGI have the advantage that they can continue running when the web page is already loaded in the browser. With CGI programs this is not possible. The only way to change/modify data that is displayed by a program using CGI is to restart the program. This could be triggered by another user event, or as it is the case in programs like `mrtg`, by a periodic refresh of the whole page[1](equivalent to the user pressing the “Reload” button of the web browser).

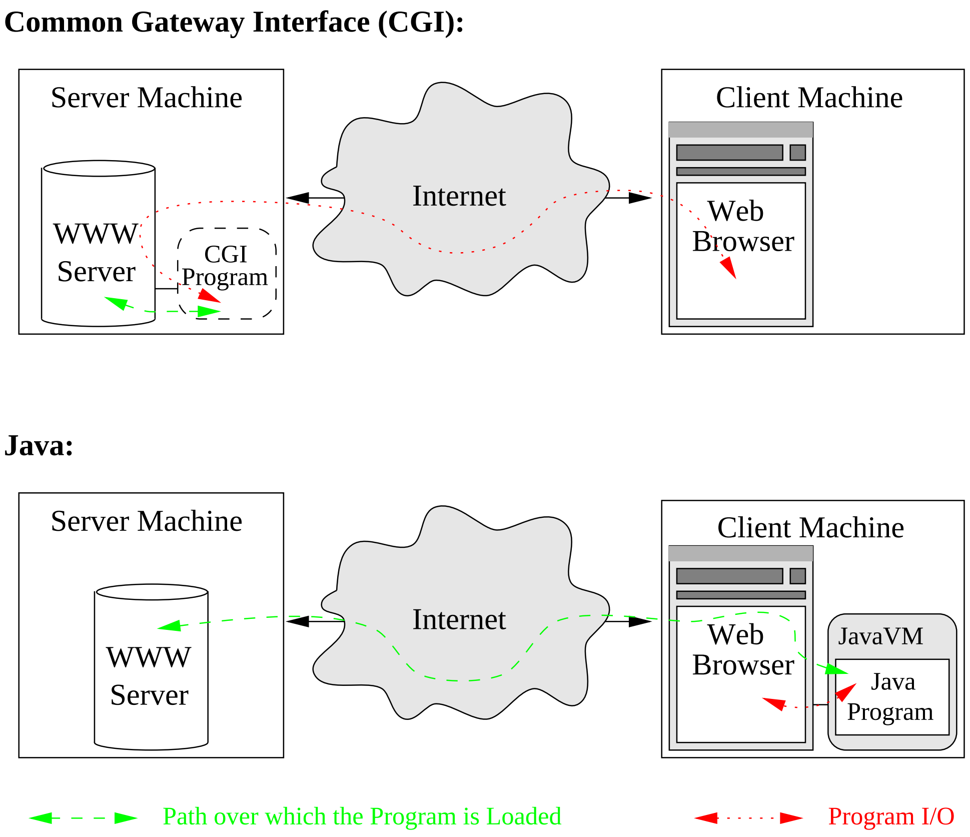


Figure 4.4: Conceptual Difference between Java and CGI Programs

[1] This can be acomplished by using the REFRESH tag in newer versions of Netscape. Note that not all web browsers are guaranteed to support this feature.

4.3 Writing Network–Management Applications for the World Wide Web

To write applications that use the SNMP protocol for network management, different possibilities exist as well. The `mrtg` product we presented in 1.2.1 uses the *Perl5 library for SNMP* by Simon Leinen[2] which itself is also written entirely in Perl. The Perl language [21] is a scripting language which is interpreted. It is very powerful to write short administrative scripts for management applications. However, the Perl library offers only a limited subset of SNMP commands. The only SNMP operations currently supported are “`GET`” and “`GET-NEXT`”. This means that one can neither set variables in an agent, nor receive or generate SNMP traps using these routines. Nevertheless, for solutions like `mrtg`, Perl5 with this library is an ideal platform, since the code is quite easy to maintain and a CGI program is perfectly capable of handling the task to generate a static graph of SNMP counters. For this particular application it is also of advantage that C programs can be easily used on the server side. Newer versions of `mrtg` use a C program to maintain the data files in order to speed up the process. Implementing this kind of application with Java would be quite inefficient.

With Java, the complete mechanisms of SNMPv2 are available using special class libraries. SNMP support is not included in the Java Development Toolkit (JDK) Version 1.0. Sun is promoting the Java Management API (JMAPI)[3]. However, when this work was started, JMAPI had not yet been released and today it is still in a beta test state. An alternative to JMAPI is provided by AdventNet: The “*Advent SNMPv2c Package*”[4] is a class library providing the complete set of functions necessary to write an SNMP application in Java. Advent uses this class library in their “*Advent Web NMS*” product, which can be used to interactively build SNMP ap-

[2] The Perl5 SNMP library is available at `http://www.switch.ch/misc/leinen/snmp/perl/`

[3] Sun provides information on the current state of the JMAPI project at `http://www.javasoft.com/products/JavaManagement/NTR.html`

[4] Documentation for the Advent products and the class libraries are available on the WWW at `http://www.adventnet.com`

plications by assembling pre–built components they provide. The library is widely in use and has shown to be suitable for a broad variety of applications.

For our application, the real–time analysis of traffic flows on a network, CGI programs with static output obviously are not suitable. Therefore we decided to write a Java application based on the AdventNet libraries.

In the following sections, we will describe the Java architecture and how SNMP applications are implemented in this architecture in detail.

4.4 Java

4.4.1 Introduction to Java

The Java language is an object–oriented programming language developed by Sun Microsystems. Modeled after C++, Java was designed to be simple, small and portable across platforms and operating systems, not only at the (compiled) binary level but also at the source level. Note that Java language has nothing to do with "JavaScript", a script language developed by Netscape that can be included in HTML pages and which can be executed by web browsers.

The Java language has been formally specified in the "Java Language Specification" [12] in 1996. Only recently the new version 1.1 of the Java Development Toolkit (JDK) has been released with which some minor changes have been made tho the language specification. This new version is documented in [18].

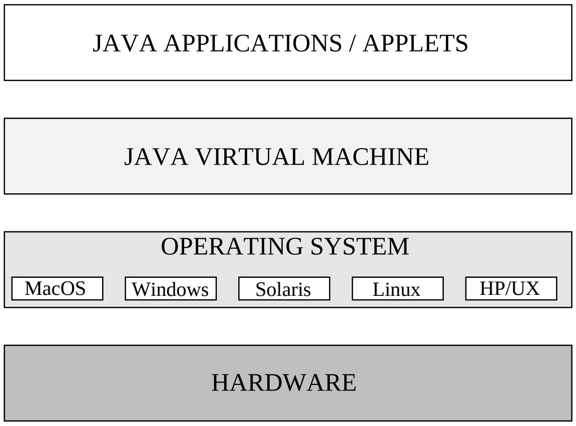


Figure 4.5: Java Execution Environment

Java sourcecode is translated into Java bytecode using the Java compiler `javac`. As mentioned, the format of the bytecode is the same on all implementations of the "Java Virtual Machine" (VM). This format, as well as the command set of the virtual machine, is specified in [24]. The VM interprets the byte code and executes it. In the future, Sun is planning to provide a special hardware chip that can directly execute the Java bytecode, but today even the special “Java Stations” Sun is vending are still based on bytecode interpreters.

Figure 4.5 shows the Execution Environment. The Java VM has already been ported to a large number of platforms. It is nowadays included in the Netscape web browser as well as the Microsoft Internet Explorer and therefore available on almost every desktop PC. Ports exist as well for all major UNIX platforms.

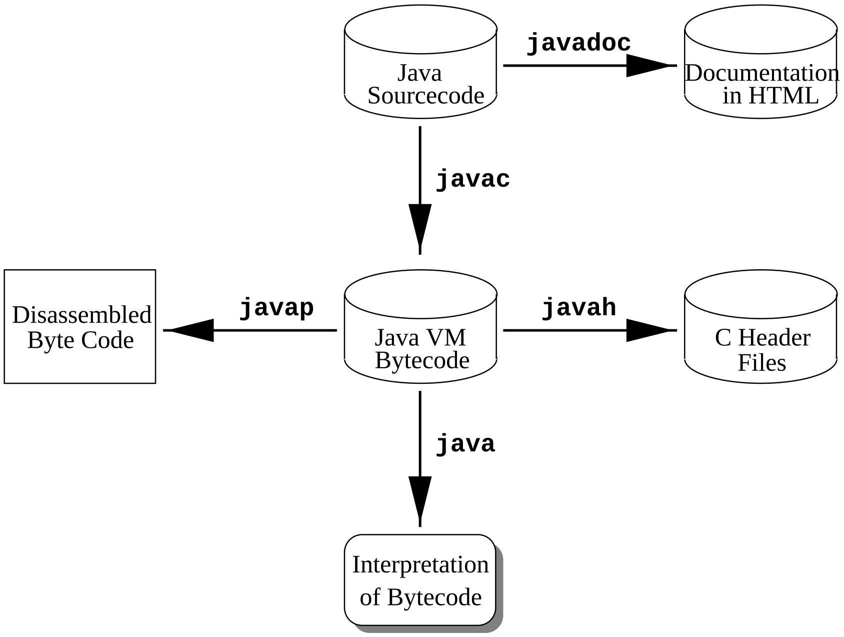


Figure 4.6: The Java Development Environment

The Java language itself is very compact and can therefore be learned easily. The important part of the JDK however is the included set of class libraries. Those libraries provide a common subset of methods and objects that can be used to write portable programs in Java. In particular, they include the “Abstract Windowing Toolkit” (AWT) that provides a common set of objects and methods for writing user interfaces. Those methods use the window manager / graphical frontend of the platform the code is running on.

Also included in the JDK are some additional tools like the `jdb` Java debugger, a bytecode disassembler and the "appletviewer". Sun also provides the "HotJava" web browser, which is completely written in Java. This browser can be used to display Java–enhanced web pages.

To summarize, the important elements of the Java environment are:

- The language specification.
- The bytecode compiler.
- The virtual machine that interprets the bytecode at runtime.
- A set of class library APIs.
- Implementations of the class libraries specific to the target machine.
- A runtime environment in which the interpreter, bytecode verifier, class loader, etc. run, also specific to the target machine.
- Other development tools such as a debugger, a disassembler and an appletviewer for testing applets outside of a web browser.
- Finally, there is also a web browser written in Java called HotJava.

Java Applets vs. Java Applications

From the users point of view, Java applications can be compared to compiled C programs since they are started from a command line just like any compiled program would be started. However, there is a major difference: Java applications, as well as applets, are interpreted. Applications are started on the command–line by calling the Java Interpreter with the name of the application as an argument. Applets, in contrary to applications, are small programs which can be included in web pages and run inside the user's browser. Alternatively, applets can be run in the appletviewer that comes with the JDK, which has advantages for debugging.

4.4.2 The Java Security Concept

Since a Java–enabled Web browser allows to embed executable Java code in a Web page, which can be downloaded across the net and run on any client machine, security is a critical concern. Users can download Java applets with exceptional ease — sometimes without even knowing it. This exposes Java users to a significant amount of risk.

For our planned SNMP applet, we will have to use the networking functions from the Java class libraries. In order to use them, it is important to understand the security concept Java uses, since networking functions are extremely dependent on the security model.

The Java security management distinguishes three different classes of Java applets. This is necessary because there is a clear difference between programs that are installed on a local machine — and therefore will always be given a certain trust that they are not "evil" — and programs that the user gets over the network. When things downloaded over the network are just text and data there is no security issue. However, downloading a Java program and executing it on the local machine is of course very much a security issue. Unless measures are taken to prevent it, a malicious (or buggy) applet could delete files, post confidential data it reads from the hard disk over the network or do other nasty stuff.

The measures that have been taken to prevent those things mostly consist of providing applets loaded over the network with a restrictive "sandbox" environment in which they run. Since not all applets are loaded over the network, and locally stored programs may be trusted more, the Java security management distinguishes between the following security classes which are given different permissions to use the resources of the machine:

Applications can connect anywhere they like, unless a new security manager which limits connections is installed. However, applications cannot run in a web browser and are therefore not interesting for us, except for the reason that they can connect any other system on the network.

Untrusted Applets are all applets that are loaded over the network. Since code that has been loaded via the network is potentially dangerous, this code is executed in a “sandbox” environment where it has only limited access to the machine the web browser is running. Especially networking capabilities are being restricted to avoid the applet communicating sensitive information to the outside.

Trusted Applets are loaded from the local machine. For trusted applets, some of the restrictions may be relaxed by the browser. They are nevertheless not granted all the privileges that applications get.

Intermediate applet security policies are also possible. An applet viewer could be implemented that would place fewer restrictions on applets loaded from an internal network (a so called “intranet”) than on those loaded from the Internet.

The restrictions the “sandbox” environment is putting on an untrusted applet include the following:

- It can’t load libraries or define native methods.
- It can’t ordinarily read or write files on the host that’s executing it.
- It can’t make network connections except to the host that it came from.
- It can’t start any program on the host that’s executing it.
- It can’t read every system property.
- Windows that an applet opens up look different than windows that an aplication opens up.

Since we want to write an applet that will run inside a web browser anywhere on our network, we have to live with those restrictions for untrusted applets. This imposes some problems, especially with SNMP, since here we forceably will need to establish network connections to SNMP agents. Since those agents are (usually) not running on the host we loaded the applet from, this imposes a major design problem here. Fortunately, AdventNet provides a solution for this problem, which we will see later.

The networking restrictions do not only depend on the source the applet is loaded from. There are also differences depending on the type of proxy server that might be located between the client browser and the web server as well as the actual implementation of the environment the applets are loaded into. Additionally, the restrictions do also depend on the kind of networking operation that is attempted. Connections to arbitrary network sockets are not routed through HTML proxies, therefore they are to be treated differently than so called “URL connections”, which are used to fetch information from web or ftp servers and may transparently access HTTP proxies.

Table 4.1 summarizes where untrusted applets are allowed to connect using `java.net.Socket` and table 4.2 summarizes where they are allowed to connect to when using `java.net.URLConnection` (to communicate with WWW/ftp servers etc.).

For SNMP applications, URL connections cannot be used. We need direct access to the SNMP port of the agent that is to be queried. As it can be seen from Table 4.1, this connectivity can only be achieved to the web server itself, and currently *only if no HTTP proxy is used.*

The solution that Advent proposes as a workaround for this problem is called “*SNMP Applet Server (SAS)*”. This server allows the applet to send and receive SNMP packets to and from any managed devices accessible from the web server host. The SAS program is a Java application and has to be run on the web server. The applet using the Advent library will then automatically communicate with the SAS application on the web server, which relays all communication to the agent that the applet

	Appletviewer	**Netscape**
No Proxy	Depending on the setting of the `appletviewer.security.mode` property, you can connect nowhere, only to the originating host, or anywhere.	Can only connect to the originating host.
SOCKS Proxy	Same as no proxy, assuming you set the `socksProxyHost` property.	No connections allowed (except under OS/2, where it's the same as with no proxy).
HTTP Proxy	In case the `appletviewer.security.mode` property is set to "none" then all connections are allowed; else no connections are allowed.	No connections allowed.

Table 4.1: Where Applets are allowed to connect to when using Sockets

	Appletviewer	**Netscape**
No Proxy	Depending on the setting of the appletviewer.security.mode property, you can connect nowhere, only to the originating host, or anywhere.	Can only connect to the originating host (= the web server).
SOCKS Proxy	Same as no proxy, assuming you set the `socksProxyHost` property.	Same as no proxy, assuming Netscape has been properly configured to use the proxy.
HTTP Proxy	Same as no proxy, assuming you set the appropriate properties (see Proxies).	Same as SOCKS Proxy.

Table 4.2: Where Applets are allowed to connect to when using URL connections

wants to communicate with. This is possible, since Java applications don't have to deal with security problems. The whole use of the SAS is completely transparent to the programmer and the user. It suffices to start the server application on the web server once, the applet can then automatically detect the TCP port that is used on the web server and all further communication will be relayed by it.

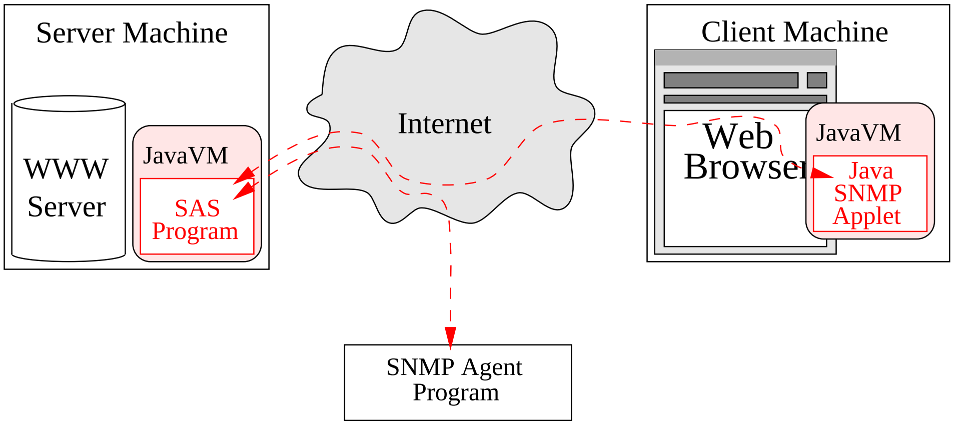


Figure 4.7: Relaying SNMP communication via the SNMP applet server (SAS)

The SNMP applet server can be used for arbitrary TCP ports, not only the SNMP port. Since this would be a potential security hole, the function of the Applet server can be limited to the forwarding of messages for the SNMP port with the command–line switch. In addition to its networking capabilities, the server also provides functionality that gives the applets limited access to files stored in a user directory on the web server.

It should be mentioned that with the recent new version of the Java JDK an additional new security mechansim is introduced: So called “signed applets”. In the future, those applets will allow the user to trust certain applets that are loaded over the network as well. For this purpose, applets can be signed using a public key mechanism. Currently, this is still under development and has no significance for us.

4.5 Technical Overview over the AdventNet SNMPv2 Java class libraries

This section briefly presents the Architecture of the AdventNet SNMPv2 Java class libraries. The package is designed to enable writing object–oriented Java applets and Java applications that use SNMP to communicate with the managed nodes.

Readers who do not want to write Java based SNMP applications or who do not know the principles of object–oriented programming may skip this section.

The package is divided into four categories. These are

1. **SNMP Variable** classes
2. **SNMP Communication** classes
3. **SNMP MIB** related classes
4. **Miscellaneous** classes, i.e. the client interface class, and the Exception sub–classes.

4.5.1 SNMP Variable Classes

The ancestor of all SNMP variable classes is an abstract class called `SnmpVar`. This class contains abstract methods for printing, ASN encoding, ASN decoding, etc.

The `SnmpVar` class has five direct sub–classes. They are:

`SnmpInt` This class is used to represent SNMP Integer syntax variables.

`SnmpNull` This class is used to represent SNMP Null variables.

`SnmpOID` The SNMP Object Identifier variable class. In addition to the usual SNMP variable class methods, this class has special constructors and methods to help interface to the MIB related classes described below.

`SnmpString` Used for SNMP Octet Strings.

The following two classes are subclasses of SnmpString:

`SnmpOpaque` Used for SNMP Opaque variable types.

`SnmpIpAddress` Used for SNMP IpAddress variable types.

`SnmpUnsignedInt` This class is not used directly but is a super–class of some of the SNMP application variable types. `SnmpUnsignedInt` has the following sub-classes.

`SnmpCounter` For SNMP Counter variable types.

`SnmpGauge` For SNMP Gauge variable types.

`SnmpTimeticks` For SNMP Timeticks variable types.

A *variable binding* is a combination of an object identifier and an SNMP variable that's commonly used in SNMP manager – agent interactions. The `SnmpVarBind` class is used for variables and methods needed for variable bindings. `ASNTypes` contains some utility functions and constants, needed in encoding and decoding SNMP variables. It is not needed for developers of applications.

4.5.2 SNMP Communication Classes

The Advent SNMP package uses the `SnmpAPI` class to manage sessions created by the user application, manage the MIB modules that have been loaded, and store some key parameters for SNMP communication, e.g. SNMP ports to be used. An SNMP application (manager or agent) often needs to manage multiple sessions on account of interacting with multiple SNMP peers. The `SnmpAPI` class has a list of sessions attached to it and monitors each of the sessions for timeouts and retransmits via a separate thread. It enables a few methods across all sessions, e.g. checking if responses have come in on any of the sessions, etc. Multiple threads can work with a single `SnmpAPI` instance. The `SnmpAPI` class must be instantiated and started to use it.

The `SnmpSession` class is used to manage a session with an SNMP peer. More than one host can be accessed via a single session, but Advent recommends to use separate sessions for hosts that are often accessed inside an application. *Each session runs as a separate thread* (primarily to do receive tasks) and provides functions to:

- Open sessions (on a particular local port if needed)
- Synchronously or asynchronously send and receive SNMP requests
- Check for responses and timeouts
- Close sessions.

An `SnmpSession` needs to be instantiated and opened before it can be used to communicate with an SNMP peer.

Interaction between the SNMP manager and the agent is done via SNMP protocol data units (PDUs). The `SnmpPDU` class will be used to provide the variables and methods to create and use the SNMP PDU.

The `SASClient` class for access to the SNMP Applet Server (SAS)

Part of the SNMP communication classes is the `SASClient` class for enabling communication through the SAS server introduced in section 4.4.2. It is not necessary to use this class directly since its use is completely transparent. It suffices to decide whether the SAS should be used when using the `open` method in the `SnmpSession` class.

In order to save applet data to a file for use later (for example to save user configuration for the next time the applet is started), the `saveFile` method in `SASClient` can be used. It saves the specified data to the specified file in the “SASusers” sub–directory on the applet host.

4.5.3 SNMP MIB Related Classes

MIB modules allow an SNMP managed agent to let users know about the structure and format of data available on the agent. The MIB modules are usually specified in a MIB module file, which needs to be parsed to understand the syntax and structure of the data available on the agent.

The `MibModule` class provides a means to parse and use the data available in a MIB module file. Each `MibModule` instance is created from a MIB module file, and you can load and unload MIB modules by creating and deleting these instances. The instance contains all the nodes of the MIB tree as well as defined traps and textual conventions. A few utility methods and variables are provided, e.g. `getNode()` to search the module for a node matching a specified OID.

The `MibModule` class makes use of a number of other classes, some of which are useful for getting additional information on the MIB module, or specific nodes in the MIB:

- The `MibMacro` class is used to parse MIB macros, and only `OBJECT-TYPE` and `TRAP-TYPE` macros are supported. Any parsing of MIB modules would instantiate the macros for the module being parsed. The `MibTrap` class is used to keep data on trap types defined in a module.
- The `MibNode` class represents a node in the parsed MIB tree. A list of instances of this class is contained in a `MibModule` and represents the MIB tree. This class may also be contained in an`SnmpOID` instance. A number of attributes and methods in the `MibNode` class are provided to simplify development of applications using the MIB definitions.
- The `LeafSyntax` class is used to represent any unique syntax, including textual conventions, that is defined in a MIB module. For example, `INTEGER (SIZE(1..5))`, would have it's own `LeafSyntax` class instance that represents this syntax. For leaf nodes in the MIB tree, the `MibNode` instance con-

tains a `LeafSyntax` reference. Thus for a MIB leaf node, `LeafSyntax` can be used to determine if a value is within the allowed range, for example.

4.5.4 Miscellaneous Classes

The following classes do not fall into the above categories, or apply across all categories.

`SnmpClient` This is an interface (Java terminology) that can be used to change default behaviour on callbacks, authentication, and where to print debugging output. To use it, the user would implement this class and set the `client` variable in the `SnmpAPI` instance.

`MibException` This is an exception class that is used to throw MIB parsing exceptions.

`SnmpException` This is an exception class to throw SNMP exceptions, e.g. decode errors.

Chapter 5

A Java Applet that works with NeTraMet Meters

The idea why we initially looked at Java was that we wanted to write a real time network analysis applet which allows to get immediate insights into the traffic on the network being monitored. In the last chapters, we introduced

a) The *flow methodology*, which allows us to reduce the amount of data collected to the minimum we really need.

b) The *NeTraMet architecture* and the *OC3MON hardware*, providing us with a platform and a standardized interface for flow measurement and analysis at high data rates.

c) The "*fluid*" application, which serves as a good example of how the collected flow information can be presented within a graphical interface

d) The AdventNet *Java SNMP class library*, which provides us with a means to write an applet that can communicate with existing network management agents.

In this chapter, we will describe how we integrate all those points into one single Java applet we called "`fluid`". This applet displays flows measured with a *NeTraMet* meter in a way like *nifty* does it. The difference is that our applet is running inside a web browser and therefore is easily usable from anywhere on the network. Currently it does not offer as many features to change the display mode, axis scaling etc. as *nifty*, but since the code is written in modular object–oriented way, it should not be too difficult to extend it.

The main goal when implementing the applet was not to write a sophisticated product but we wanted to examine whether it was possible to do this in Java.

5.1 Design of the Applet

5.1.1 The Environment

Figure 5.1 depicts the complete environment the applet will run in. First, its code is loaded from the web server into the web browser (as the green line in the figure shows). It then can communicate with the *NeTraMet* meter (which is an SNMP agent) via the SNMP applet server application. The applet server is running on the web server machine and relays all SNMP messages between the applet and the agent. The *NeTraMet* meter is configured using the standard tools (*NeMac*, *nifty*) and can be used to meter any of the network segments that the host it is running on has access to.

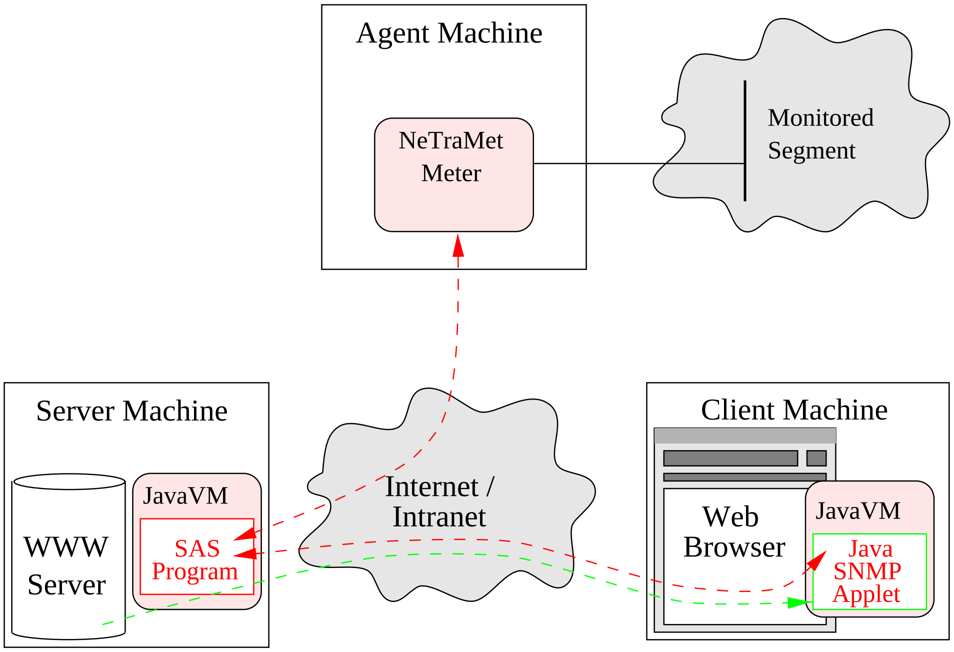


Figure 5.1: Environment in which the Applet is running

5.1.2 The Architecture of the Applet

Within the IETF architecture described in section 3.2.1, the applet implements the functionality of the meter reader as well as the analysis application. It does *not* implement the meter managers functionality, as *nifty* for example does. This could be added in later versions, but for our study we treated it with lower priority. The manager functionality can completely be taken over by *NeMaC*.

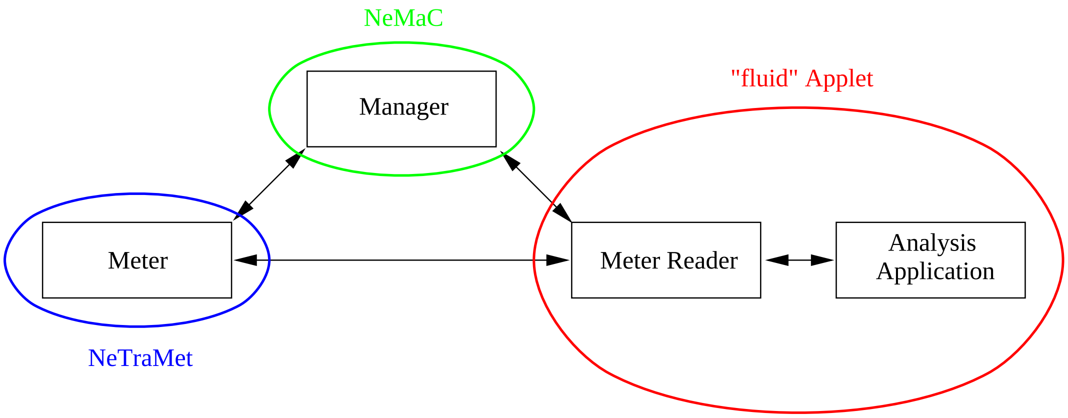


Figure 5.2: How the “fluid” Applet fits in the IETF RTFM Architecture

Figure 5.2 illustrates which parts of the RTFM architecture are implemented within the “fluid” applet. Since the “Manager” functionality is not implemented in Java, the *NeMac* program is used for uploading of the ruleset file to the meter. This is an advantage of the RTFM architecture: One can implement one application after another, independently from each other. A Java implementation of the manager functionality could for example be done as a next development step.

5.1.3 Organization of the Java Code

The Java code for the Applet is split into multiple parts. The main program is contained in the file “`fluid.java`”. This file includes the `fluid` class, which extends the `applet` class and is run in the first place after the bytecode has been loaded into the browser.

The applet makes use of so called “threads”. This is very advantageous when programming SNMP communications. Using threads, one can send SNMP requests asynchronously at any time without having to wait for a reply. The response is received within the “callback” method, which is running inside a separate thread. Alternatively, one can still use synchronous communication. When programming the main SNMP communications with the *NeTraMet* meter we found this technique very convenient.

Besides the code in “`fluid.java`”, there are some other files that contain mainly the code for the user interface. Those are:

`netramet/FlowDataEntry.java` describes the data structure that is stored for each flow inside the applet.

`netramet/Ruleset.java` describes the data structure of rulesets. It is currently only used for the search of the ruleset in the meters table. Later it could be expanded to hold the whole Ruleset information in order to allow ruleset upload as well.

`StatusPanel.java` Contains the code for the display of general status information, like connection state with the meter, SNMP community name etc.

`FlowPanel.java` which represents the canvas in which the actual flow data is displayed.

`FlowFrame.java` which contains the code for the Pop–Up windows that appear when data for a particular flow is to be displayed.

Figure 5.3 gives an overview of the applets structure. As it can be seen from the figure, more than one `FlowFrame` object can be child of the `FlowPanel`. These frames contain the information about particular flows and are opened whenever the user selects one of the points inside the flow panel representing a flow.

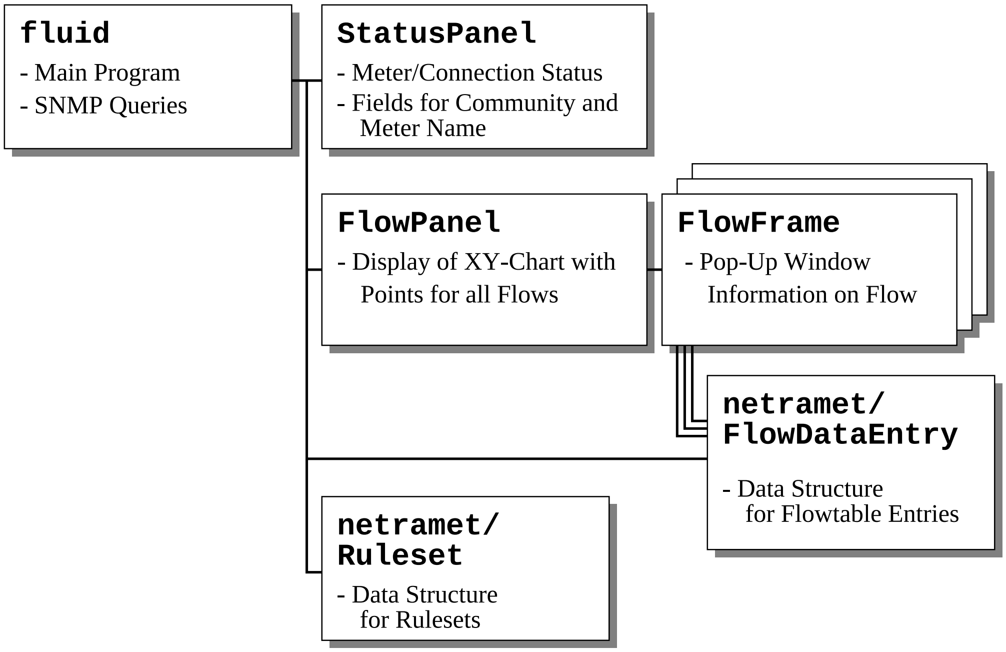

fluid
- Main Program
- SNMP Queries
StatusPanel
- Meter/Connection Status
- Fields for Community and Meter Name
FlowPanel
- Display of XY-Chart with Points for all Flows
FlowFrame
- Pop-Up Window Information on Flow
netramet/ FlowDataEntry
- Data Structure for Flowtable Entries
netramet/ Ruleset
- Data Structure for Rulesets

Figure 5.3: Structure of the Java Sourcecode

5.2 Installation and Usage

5.2.1 Preparation of the Web Server

To install the program on a web server, it suffices to put the compiled Java files (which have the extension `.class`) into the directory structure of the web server. An HTML file with an APPLET tag must then be created in the same directory. Figure 5.4 shows an example for the syntax of this tag.

```
<APPLET CODE="fluid.class"
WIDTH=500 HEIGHT=650
NAME="by Frieder Loeffler" CODEBASE=".">
<PARAM NAME=port VALUE="161">
<PARAM NAME=hostname VALUE="ksoc3mon2.rus.uni-stuttgart.de">
<PARAM NAME=community VALUE="frieder">
<PARAM NAME=MIBFILE1 VALUE="rfc1213-MIB">
<PARAM NAME=MIBFILE2 VALUE="newflowmib">
</APPLET>
```

Figure 5.4: Example of how the Applet is included into a Web Page

5.2.2 Starting the NeTraMet meter

Before the applet can be used, a *NeTraMet* meter that is to be queried must be installed at the measurement point. If the OC3MON NeTraMet is to be used, this must be started from the DOS command line as follows, where “frieder” in this example is the SNMP write community name for write–access to the meter.

```
set DPMIMEM=MAXMEM 8192
set HOST_CLOCK_RATE=166e6
mode co80,50
ntmoc3 -5 -wfrieder
```

The environment variables that have to be set are used by the OC3MON code and are not relevant for operation of *NeTraMet*.

In case of the UNIX version of *NeTraMet* being used, the meter would have to be started as user "root" with the following command:

```
NeTraMet -wfrieder
```

Once the meter is started, it will analyze all packets that it sees and will try to aggregate them to flows as defined by the rulesets that are uploaded to it.

5.2.3 Uploading Rulesets with a manager application

Before the applet can be used, the meter must be programmed with the flow specification it should use. For this purpose, the manager functionality of the programs that come with *NeTraMet* can be used. Supposed that the meter is running on host `ksoc3mon`, the SNMP write community name is `frieder` and the rulesets to use would be specified in a file called `myxrules`, the rulesets could be uploaded to the meter either by calling *nifty* as

```
nifty -c 120 -r myxrules ksoc3mon frieder
```

or by calling *NeMaC* as:

```
NeMaC -c 120 -r myxrules ksoc3mon frieder
```

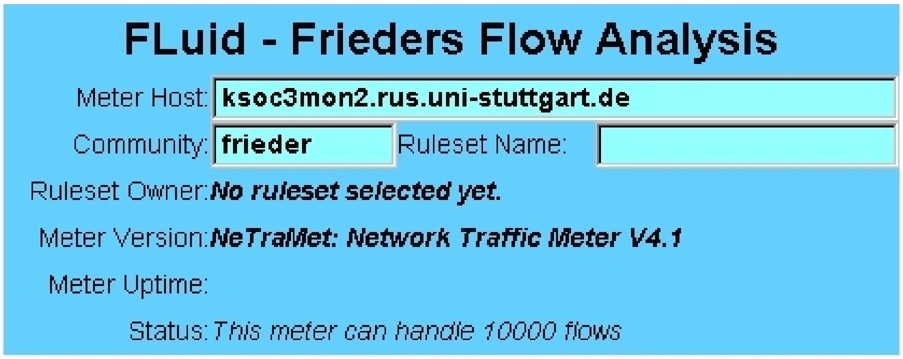
An example for a ruleset file can be found in Appendix A. Within the ruleset file it is possible to change the letters or symbols that are used to depict the flows of different kinds. Please consult the *nifty* manual[1] for details.

[1] The manual is part of the *NeTraMet* distribution

5.2.4 Using the Applet

Once the web page is loaded, the applet will be started automatically. To use it, the user first has to enter the **hostname** of the machine the NeTraMet meter is running on. (A default for the hostname can be set by specifying the “hostname” parameter in the tag, as in the example). Additionally, the SNMP **community name** must be specified in the corresponding field.

Once those two parameters are entered, the applet will connect to the remote *NeTraMet* meter. If it succeeds, it will display the *NeTraMet* Version number and maximum number of flows in the “Meter Version” and “Status” fields. If it does not succeed, the message “No Meter” will be displayed in the status field. Figure 5.5 shows what the status fields in the applet should look like after succesfully connecting to the meter.



FLuid - Frieders Flow Analysis

Meter Host: **ksoc3mon2.rus.uni-stuttgart.de**

Community: **frieder** Ruleset Name:

Ruleset Owner: ***No ruleset selected yet.***

Meter Version: ***NeTraMet: Network Traffic Meter V4.1***

Meter Uptime:

Status: *This meter can handle 10000 flows*

Figure 5.5: Status frame of the “fluid” applet after succesfully connecting to the meter

After the connection has been successfully established, the user has to enter the name of the ruleset that was uploaded with *NeMaC* or *nifty*. In case of the ruleset shown in Appendix A, the name would be “2” (the name appears after the `SET` keyword in the ruleset file).

Once the name of the ruleset has been typed into the “ruleset name” field of the applet and the return key is pressed, the applet tries to find that ruleset in the meters `flowRuleSetInfoTable`. If it suceeds, it registers itself in the `flowReaderInfoTable` of the meter as a meter reader that uses this set. The name of the Ruleset owner will then be shown in the "Ruleset Owner" field of the status frame.

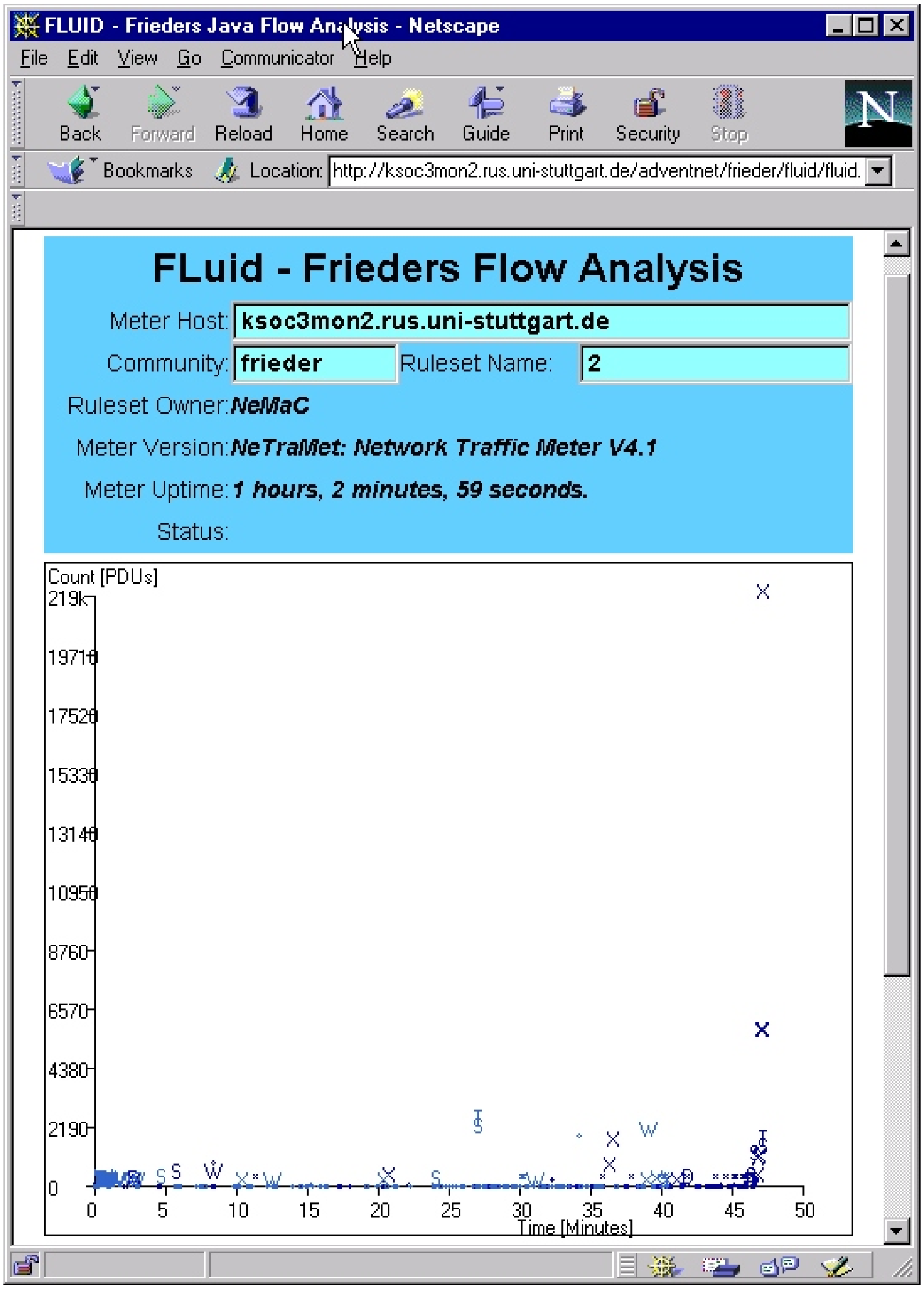
Thereafter, no more user intervention is necessary. The applet will now try to query the meter in regular intervals to read the `flowDataTable`. During the queries, the flow identifier of the flow that is currently beeing analysed will be shown in the "Status" field. On heavily loaded networks, this process can take a while.

Running the applet in the JDK 1.0.2 appletviewer on a Pentium–166 with the Linux–2.0 operating system, it took about 10 seconds to transfer the data records for 300 flow records from the meter to the applet. The implementation already uses "`GETBULK`" requests on the `flowDataPackageTable` to transfer the packaged record data. Further improvements in speed can therefore only be achieved by using a faster Java environment (for example a just–in–time compiler might help). This shows that the transfer time for the flow table is a critical factor when implementing RTFM applications in Java.

After the whole flow table has been transferred, the applet will display all flows in its table inside the XY–graph. Each flow is represented with either one character or a symbol, depending on the definitions used in the ruleset file. The same syntax as in *nifty* is used here, so that ruleset files developed for use with *nifty* can be used without modification. It is even possible to upload a ruleset file only once and display the information with *fluid* and *nifty* at the same time.

After some minutes, the applets display area looks like depicted in Figure 5.6. The Y axis depicts the "active time", i.e. how long the period was in that packets have been detected for that flow by the meter. On the Y axis, the applet displays the number of PDUs that have been received by the meter for that particular flow. Currently, the number of received and sent PDUs is added before the flow is displayed.

Once no more data for a flow is being received, i.e. the flow is no longer "current", the color in which its letter or symbol is being displayed will slowly fade to white.



Location: http://ksoc3mon2.rus.uni-stuttgart.de/adventnet/frieder/fluid/fluid.

FLuid - Frieders Flow Analysis

Meter Host: ksoc3mon2.rus.uni-stuttgart.de

Community: frieder Ruleset Name: 2

Ruleset Owner: *NeMaC*

Meter Version: *NeTraMet: Network Traffic Meter V4.1*

Meter Uptime: *1 hours, 2 minutes, 59 seconds.*

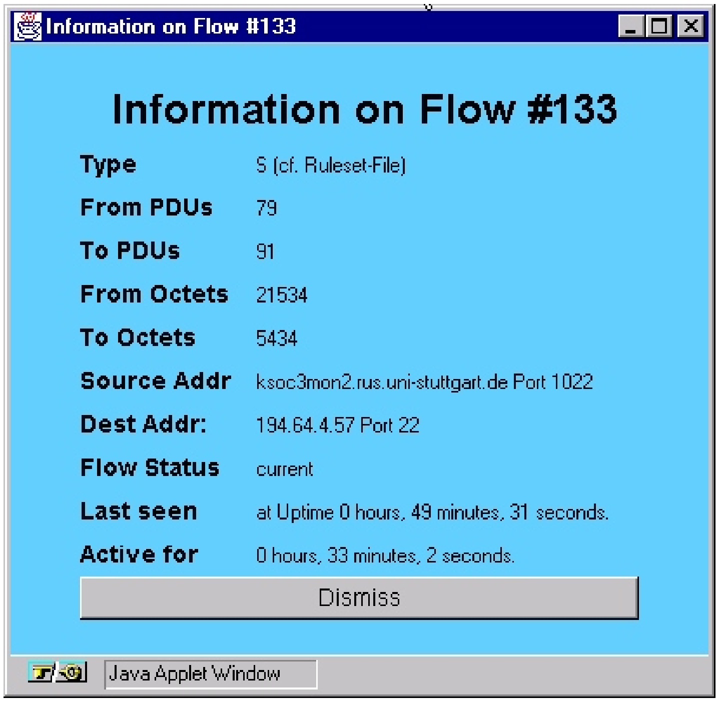
Status:

Figure 5.6: The “fluid” applet displaying information about flows

In the current implementation, a flow will be deleted from the display once it has been inactive for more than 1000 seconds.

5.2.5 Getting more detailed Information

In order to get additional information about a particular flow (for example to find out which machines are generating excess traffic on the network), the user can select each of the characters or symbols by clicking with the mouse on it. A window as depicted in figure 5.7 will then open up.



Information on Flow #133

Type	S (cf. Ruleset-File)
From PDUs	79
To PDUs	91
From Octets	21534
To Octets	5434
Source Addr	ksoc3mon2.rus.uni-stuttgart.de Port 1022
Dest Addr:	194.64.4.57 Port 22
Flow Status	current
Last seen	at Uptime 0 hours, 49 minutes, 31 seconds.
Active for	0 hours, 33 minutes, 2 seconds.

Figure 5.7: A “fluid” window containing information about a particular flow

The title line of this window is used to display the flow number, which together with the time when the flow was first seen can be used as a unique flow identifier. Below, the flow kind, as specified in the ruleset file, can be determined. In our example, the letter “S” is used for all “ssh” (secure shell) flows, i.e. flows that use the TCP port number 22.

The “From PDUs” and “To PDUs” lines are self–explaining. They give information about the number of PDUs transferred in each direction. In the same way the two following lines inform about the number of octets that were transferred. By dividing the number of octets by the number of PDUs one can easily see the average size of a PDU in each direction of the flow.

The two following lines show the source and destination IP[2] address. From the destination address field in the example one can indeed see that the flow depicted here was a secure shell (ssh) flow, since the port number used on the destination machine was the secure shell port (22). Only the source address is displayed in non–numeric form because of a bug in the current Java environment. This bug restricts the access to the name server – the socket connection that would be needed in order to look up the host name of the destination machine is refused by the security manager.

The “Flow Status” field is also self–explaining. It can either contain the value “current” for flows for which data has been seen during the last measurement inteveral of the meter, or “inactive” when no data has been seen.

The last two lines are used to display exact time information for the flow. By calculating the difference between the two time values, one can determine the time when the flow was first seen.

Whenever the meter is queried and new information for a flow for which an information window is still open is received, this window is updated. This allows the user to permanently monitor the counts for one particular flow. As an example, the network manager could use this feature to select the traffic flow for a given application he suspects to misbehave. He would then just leave the flow information window open and could verify how much data this application is generating in both directions.

[2] Note that currently the applet does not support any other network protocols than TCP/IP

Chapter 6

Further Work

6.1 The Fluid Applet and the RTFM Working Group

6.1.1 The Fluid Applet

The most important features that are still missing from the applet are certainly the ones already implemented in *nifty*:

- Possibilities to choose different criteria for the Y–axis, for example number of octets transferred, byte rate, packet rate etc.

- Possibitities to change the scaling of both axes. Currently an automatic algorithm is used for scaling. This algorithm always scales the axis in a way so that the maximum value can still be displayed.

Besides those rather cosmetical additions, an interesting possibility for further work would be to integrate the whole applet into an object oriented environment like the AdventNet product line of Network Management products. The applet code could then be used as one module for building complex network monitoring applications. At the same time, the existing user interface components could easily be used for displaying graphs over all kinds of counters. One could for example imagine that the user picks one particular flow out of the XY–graph that is displayed and then decides to make a graph over the number of bytes or packets transferred for this flow. This could be used to monitor just a particular pair of hosts activity over longer periods of time. Currently such fine granularity is not available with the standard network monitoring and management software.

Other things to add or improve might be to allow graphing for bidirectional traffic criteria, i.e. two points in the charts or 3–D charts displaying the sent and received PDUs or number of octets and not only the total, as it is currently done.

6.1.2 Ongoing Developments within the RTFM Working Group

On the 39th IETF meeting (Munich, August 1997), Nevil Brownlee presented some interesting new methodologies for using `NeTraMet` to measure not only volume (using counters) but other parameters with *distributions*. In the recent beta versions, he added features that allow the measurement of per flow packet interarrival times and turnaround times.

Measurement methodologies that were previously applied to the whole network traffic will become applicable to selected flows. Those flows can be specified by using ruleset files as it is done within NeTraMet right now since distributions will seamlessly be added to the current implementation. Currently, the RTFM working group is trying to find “interesting” new attributes like the ones mentioned above to be added to the MIB.

6.2 Open Questions

There are many questions still open and to study — both formally and with respect to the application field.

So far, with flows declaration preceeded their application, i.e. we were using flows statically.

It is an interesting question whether there are situations where it could be of interest to change the flow specification — e.g. aggregation level, time out parameter etc. — *dynamically* in response to changing network situations.

Typical questions with respect to the *application field* would be:

- Can we use a flow–based instant characterization of the network as a parameter in a RSVP–policy function?
- How would we relate a static or dynamic flow picture of an IP 0ver ATM network to determine e.g. the holding times for ATM SVCs?

The flow methodology and the applications presented in this thesis could be used in further studies on such questions.

Chapter 7

Conclusions

This thesis focused on the third of the three “flow” contexts “Resource Reservation”, “Switching” and “Measurement and Analysis”. From the comparison of different tools and strategies for network monitoring, the IETF Realtime Traffic Flow Measurement model with its prototype applications (*NeTraMet, NeMaC* and *nifty*) showed to be the most advanced one. It can already be used on many kind of network technologies, most interesting certainly the version for the OC3MON. This is insofar an important contribution as it allows to use the OC3MON hardware to build low–cost customized network measurement environments for monitoring on ATM OC3 links using a standardized interface. It can only be hoped that manufacturers of networking equipment will soon begin to implement the RTFM meter functionality into their equipment. The standardization process within the IETF has reached a state where this would be the next logical step.

The work on the Java based traffic flow analyser is not yet finished by far. The contribution has reached a state in which it can be used to show that Java is useful for this kind of application. The language has shown to be suitable for the development of such network management applications, not only because the object–oriented development is helpful when adapting the program to local requirements. The contribution has also shown that the speed of the Java Virtual Machine on standard desktop PCs is high enough for such flow–based analysis applications. For the

RTFM working group, the Java applet is important, since — according to the IETF rules — the architecture needs to be implemented in two genetically independent forms before it can become a standard. Our Java–based implementation contributes to the standardization process by providing a genetically independent meter reader.

Having the thesis focused on Measurement and Analysis aspects of flows, we feel this to be necessary in order to make the flow paradigm eventually applicable to Reservation and Switching.

Appendix A

Ruleset file for “fluid”

```
# Rulesetfile for the "fluid" applet and the "nifty" flow analyzer.
# Derived from Nevil Brownlee’s "nifty" ruleset file.
#
# If your meter hostname was "ksoc3mon" and your SNMP community name
# was ‘‘frieder’’, you would upload this file using either
# "nifty" or "NeMaC" to a NeTraMet meter with one of the following commands:
#
#    nifty -c 120 -r myxrules ksoc3mon frieder
#    NeMaC -c 120 -r myxrules ksoc3mon frieder
#
# (c) Siegfried Loeffler 07/97
#
SET 2
#
RULES
  SourcePeerType & 255 = dummy:  Ignore, 0;
  SourcePeerType & 255 = IP:     Pushto, IP_pkt;
  SourcePeerType & 255 = 4:   Pushto, IP_pkt;
  SourcePeerType & 255 = Other:  PushToAct, other_pkt;
#
  Null & 0 = 0:   GotoAct, Next;  # Not IP or Other
  FlowKind & 255 = 3:  PushtoAct, Next;  # Plot as SQUARE
  SourceInterface & 255 = 0:  PushPkttoAct, Next;
  SourcePeerType  & 255 = 0:  CountPkt, 0;
#
other_pkt:  # We want to know ethertype/LSAP (in source/dest Peer)
  FlowKind & 255 = 3:  PushtoAct, Next;  # Plot as SQUARE
  SourceInterface & 255 = 0:  PushPkttoAct, Next;
  SourcePeerAddress  & 255.255 = 0: PushPktToAct, Next;
  DestPeerAddress    & 255.255 = 0: CountPkt, 0;
```

```
#
IP_pkt:
  SourceTransType & 255 = tcp:    Pushto, tcp_udp;
  SourceTransType & 255 = udp:    Pushto, tcp_udp;
  SourceTransType & 255 = icmp:   GotoAct, c_icmp;
  Null & 0 = 0:   GotoAct, Next;  # Not TCP or UDP
  SourceTransType  & 255 = 0:  PushPkttoAct, Next;
  FlowKind & 255 = 3:  PushtoAct, count_IP;  # Plot as SQUARE
#
tcp_udp:
  SourceTransAddress & 255.255 = domain:   Retry, 0;  # Want WKP as dest
  SourceTransAddress & 255.255 = 22:       Retry, 0;
  SourceTransAddress & 255.255 = 79:       Retry, 0;
  SourceTransAddress & 255.255 = ftp:      Retry, 0;
  SourceTransAddress & 255.255 = ftpdata:  Retry, 0;
  SourceTransAddress & 255.255 = gopher:   Retry, 0;
  SourceTransAddress & 255.255 = 143:      Retry, 0;
  SourceTransAddress & 255.255 = 513:      Retry, 0;
  SourceTransAddress & 255.255 = 137:      Retry, 0;  # NETBIOS Name Service
  SourceTransAddress & 255.255 = 138:      Retry, 0;  # NETBIOS Datagram
  SourceTransAddress & 255.255 = 139:      Retry, 0;  # NETBIOS Session
  SourceTransAddress & 255.255 = nntp:     Retry, 0;
  SourceTransAddress & 255.255 = 2049:     Retry, 0;
  SourceTransAddress & 255.255 = ntp:      Retry, 0;
  SourceTransAddress & 255.255 = 110:      Retry, 0;
  SourceTransAddress & 255.255 = 515:      Retry, 0;
  SourceTransAddress & 255.255 = smtp:     Retry, 0;
  SourceTransAddress & 255.255 = snmp:     Retry, 0;
  SourceTransAddress & 255.255 = 1080:     Retry, 0;  # UA socks gateway
  SourceTransAddress & 255.255 = telnet:   Retry, 0;
  SourceTransAddress & 255.255 = www:      Retry, 0;
  SourceTransAddress & 255.255 = 3128:     Retry, 0;  # Squid cache
  SourceTransAddress & 255.255 = 3130:     Retry, 0;  # Squid cache control
  SourceTransAddress & 255.255 = 8080:     Retry, 0;  # UA WWW proxy
  SourceTransAddress & 255.255 = 6000:     Retry, 0;
#
  DestTransAddress & 255.255 = domain:     GotoAct, c_domain;
  DestTransAddress & 255.255 = 22:         GotoAct, c_ssh;
  DestTransAddress & 255.255 = 79:         GotoAct, c_finger;
  DestTransAddress & 255.255 = ftp:        GotoAct, c_ftp;
  DestTransAddress & 255.255 = ftpdata:    GotoAct, c_ftpdata;
  DestTransAddress & 255.255 = gopher:     GotoAct, c_gopher;
  DestTransAddress & 255.255 = 143:        GotoAct, c_imap;
  DestTransAddress & 255.255 = 513:        GotoAct, c_login;
  DestTransAddress & 255.255 = 137:        GotoAct, c_netbios;  # Name
  DestTransAddress & 255.255 = 138:        GotoAct, c_netbios;  # Datagram
  DestTransAddress & 255.255 = 139:        GotoAct, c_netbios;  # Session
  DestTransAddress & 255.255 = nntp:       GotoAct, c_news;
  DestTransAddress & 255.255 = 2049:       GotoAct, c_nfs;
  DestTransAddress & 255.255 = ntp:        GotoAct, c_ntp;
  DestTransAddress & 255.255 = 110:        GotoAct, c_pop;
  DestTransAddress & 255.255 = 515:        GotoAct, c_printer;
  DestTransAddress & 255.255 = smtp:       GotoAct, c_smtp;
  DestTransAddress & 255.255 = snmp:       GotoAct, c_snmp;
  DestTransAddress & 255.255 = 1080:       GotoAct, c_socks;  # UA socks
  DestTransAddress & 255.255 = 3130:       GotoAct, c_squid_control;
  DestTransAddress & 255.255 = 3128:       GotoAct, c_squid_data;
  DestTransAddress & 255.255 = telnet:     GotoAct, c_telnet;
  DestTransAddress & 255.255 = www:        GotoAct, c_www;
  DestTransAddress & 255.255 = 8080:       GotoAct, c_www;  # UA WWW proxy
  DestTransAddress & 255.255 = 6000:       GotoAct, c_xwin;
#
  Null & 0 = 0:  GotoAct, c_tcp_udp;  #  ’Unusual’ port
#
c_domain:
  FlowKind & 255 = ’D’:  PushtoAct, count_IP;
c_ftp:
c_ftpdata:
  FlowKind & 255 = ’F’:  PushtoAct, count_IP;
c_imap:
  FlowKind & 255 = ’I’:  PushtoAct, count_IP;
c_netbios:
  FlowKind & 255 = ’B’:  PushtoAct, count_IP;
c_news:
  FlowKind & 255 = ’N’:  PushtoAct, count_IP;
c_pop:
  FlowKind & 255 = ’P’:  PushtoAct, count_IP;
c_smtp:
  FlowKind & 255 = ’M’:  PushtoAct, count_IP;
#c_socks:
#  FlowKind & 255 = ’S’:  PushtoAct, count_IP;
c_ssh:
  FlowKind & 255 = ’S’: PushtoAct, count_IP;
c_squid_data:
  FlowKind & 255 = ’C’:  PushtoAct, count_IP;
c_squid_control:
  FlowKind & 255 = ’c’:  PushtoAct, count_IP;
c_telnet:
  FlowKind & 255 = ’T’:  PushtoAct, count_IP;
c_www:
  FlowKind & 255 = ’W’:  PushtoAct, count_IP;
c_xwin
  FlowKind & 255 = ’X’:  PushtoAct, count_IP;
#
c_finger:
  FlowKind & 255 = ’f’:  PushtoAct, count_IP;
c_gopher:
c_login:
```

```
c_nfs
c_ntp:
c_printer:
c_snmp:
c_socks:
#
c_tcp_udp:
  Null & 0 = 0:   Goto, Next;  # Not a well-known TCP or UDP port
  SourceTransType & 255 = tcp:  GotoAct, c_tcp;
  Null & 0 = 0:   GotoAct, c_udp;
c_udp:
  FlowKind & 255 = 2:  PushtoAct, count_IP;  # Plot as PLUS
c_tcp:
  FlowKind & 255 = 1:  PushtoAct, count_IP;  # Plot as DIAMOND
c_icmp:
  FlowKind & 255 = '*': PushtoAct, count_IP;
#
count_IP:
  SourceInterface & 255 = 0:  PushPkttoAct, Next;
  SourcePeerAddress  & 255.255.255.255 = 0:  PushPkttoAct, Next;
  DestPeerAddress    & 255.255.255.255 = 0:  PushPkttoAct, Next;
  SourceTransAddress & 255.255         = 0:  PushPkttoAct, Next;
  DestTransAddress   & 255.255         = 0:  CountPkt, 0;
#
#
#FORMAT
#  FlowRuleSet FlowIndex FirstTime "  "
#  SourcePeerType "  "
#  SourcePeerAddress DestPeerAddress "  "
#  SourceTransAddress DestTransAddress "  "
#  ToPDUs ToOctets "  " FromPDUs FromOctets "   "
#  FirstTime LastTime
#
# end of file
```

Appendix B

Overview over the Flow MIB

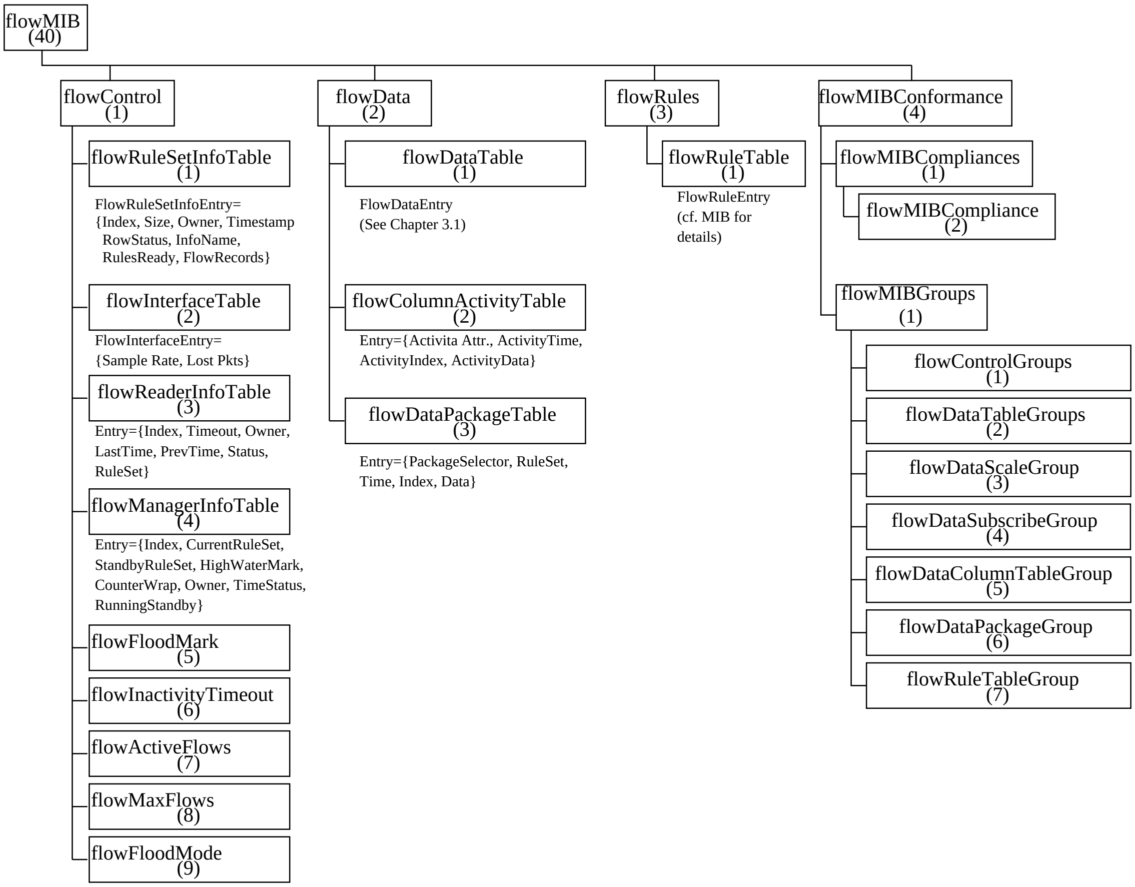


Figure B.1: Overview over the RTFM Flow MIB

Bibliography

[1] M. Acharya and B. Bhalla. A flow model for computer network traffic using real–time measurements. In *Second International Conference on Telecommunications Systems, Modeling and Analysis*, March 1994.

[2] S. Bostock. SNMP over IPX. Request for Comments (Experimental) RFC 1420, Internet Engineering Task Force, March 1993.

[3] Daniel Freedman, Chris Metz, Jaap Burger (IBM) Brian Dorling. *Internetworking over ATM*. Prentice–Hall, 1996.

[4] Nevil Brownlee. Traffic flow measurement: Meter MIB. Request for Comments (Experimental) RFC 2064, Internet Engineering Task Force, January 1997.

[5] Nevil Brownlee. Traffic flow measurements: Experiences with netramet. Request for Comments (Informational) RFC 2123, Internet Engineering Task Force, March 1997.

[6] Kimberly C. Claffy. *Internet traffic chararcterization*. PhD thesis, University of California, San Diego, 1994.

[7] D. Hirsh, C. Mills, G. Ruth. Internet accounting: Background. Request for Comments (Informational) RFC 1272, Internet Engineering Task Force, November 1991.

[8] Martin de Prycker. *Asynchronous Transfer Mode – Solution For Broadband ISDN*. Ellis Horwood, UK, second edition, 1993.

[9] D. Estrin and D. Mitzel. An assessement of state and lookup overhead in routers. In *Proceedings of IEEE Infocom 92*, May 1992. pp. 2332-42.

[10] Juha Heinanen. Multiprotocol encapsulation over ATM adaptation layer 5. Request for Comments (Experimental) RFC 1483, Internet Engineering Task Force, July 1993.

[11] R. Jain and S. A. Routhier. Packet trains — measurement and a new model for computer network traffic. *IEEE Journal on Selected Areas in Communications*, 4(6), September 1986.

[12] James Gosling, Bill Joy, Guy Steele. The Java Language Specification (Version 1.0). Technical report, Sun Microsystems, 1996.

[13] Joel Apisdorf, K. C. Claffy, Kevin Thompson, Rick Wilder. OC3MON – flexible, affordable, high performance statistics collection. Technical report, National Laboratory for Applied Networks Research (NLANR), 1996.

[14] K. C. Claffy, H.-W. Braun, G. C. Polyzos. A parametrizable methodology for internet traffic flow profiling. *IEEE JSAC Special Issue on the Global Internet*, 1995.

[15] K. McCloghrie, M. Rose. Structure and identification of management information for TCP/IP-based internets. Request for Comments (Standard) RFC 1155 (STD16), Internet Engineering Task Force, May 1990.

[16] K. McCloghrie, M. Rose. Management information base for network management of TCP/IP-based internets: MIB-II. Request for Comments (Standard) RFC 1213 (STD 17), Internet Engineering Task Force, March 1991.

[17] F. Kastenholz. Definitions of managed objects for the ethernet-like interface types. Request for Comments (Standard) RFC 1623 (STD 50), Internet Engineering Task Force, May 1994.

[18] Ken Arnold, James Gosling. *The Java Programming Language*. Addison Wesely Longman, 1996.

[19] W. Kernighan and D. M. Ritchie. *The C programming Language (2nd ed)*. Prentice–Hall Inc., Englewood Cliffs, N. J. 07632, 1988.

[20] Larry L. Peterson, Bruce S. Davie. *Computer Networks – A Systems Approach*. Morgan Kaufmann Publishers, San Francisco, CA, 1996.

[21] Larry Wall, Tom Christiansen, Randal L. Schwartz. *Programming Perl — 2nd Edition*. O'Reilly & Associates, Inc., September 1996.

[22] M. Acharya, R. Newman-Wolfe, H. Latchman, R. Chow and B. Bhalla. Real–time hierarchical traffic characterization of a campus area network. In *Proceedings of the Sixth International Conference on Modelling Techniques and Tools for Computer Performance Evaluation*. University of Florida, 1992.

[23] M. Schoffstall, M. Fedor, J. Davin, J. Case. A simple network management protocol (SNMP). Request for Comments (Standard) RFC1157 (STD15), Internet Engineering Task Force, May 1990.

[24] Martin Lindholm, Frank Yellin. *Java Virtual Machine Specification*. Addison Wesley Longman, 1996.

[25] J. Mogul. Observing TCP dynamics in real networks. In *Proceedings of ACM SIGCOMM '91*, 9 1991.

[26] N. Brownlee, C. Mills, G. Ruth. Traffic flow measurement: Architecture. Request for Comments (Experimental) RFC 2063, Internet Engineering Task Force, January 1997.

[27] P. Newman, W. Edwards, R. Hinden, E. Hoffman, F. Liaw, T. G. Minshall. Transmission of flow labelled IPv4 on ATM data links. Request for Comments (Informational) RFC 1954, Internet Engineering Task Force, May 1996.

[28] P. W. Edwards, R. E. Hoffman, F. Liaw, T. Lyon, G. Minshall. Ipsilon flow management protocol specification for IPv4 version 1.0. Request for Comments (Informational) RFC 1953, Internet Engineering Task Force, May 1996.

[29] C. Partridge. A proposed flow specification. Request for Comments (Proposed Standard) RFC 1363, Internet Engineering Task Force, September 1992.

[30] Peter Newman, Tom Lyon, Greg Minshall. Flow labelled IP: A connectionless approach to ATM. *IEEE Infocom*, March 1996.

[31] R. Caceres, P. Danzig, S. Jamin and D. Mitzel. Characteristics of wide–area TCP/IP conversations. In *Proceedings of ACM SIGCOMM '91*, 9 1991. pp. 101-112.

[32] R. Enger, J. Reynolds. FYI on a network management tool catalog: Tools for monitoring and debugging TCP/IP internets and interconnected devices. Request for Comments (FYI) RFC1420 (FYI2), Internet Engineering Task Force, June 1993. (Obsoletes RFC1298).

[33] Rolf M. Schmid, Reto Beeler, Silvia Giordano, Hannu Flinck. IP and ATM. Position paper, ACTS AC094: EXPERT, 1996.

[34] M. Rose. A convention for defining traps for use with the SNMP. Request for Comments (Informational) RFC 1215, Internet Engineering Task Force, March 1991.

[35] M. Rose. SNMP over OSI. Request for Comments (Experimental) RFC 1418, Internet Engineering Task Force, March 1993.

[36] Marshall T. Rose. *The Simple Book – An Introduction to Management of TCP/IP Based Internets*. Prentice Hall, 1991.

[37] S. W. Handelmann, N. Brownlee, Greg Ruth. Real time flow measurement working group — new attributes for traffic flow measurement. Technical report, Internet Engineering Task Force, March 1997.

[38] William Stallings. *SNMP, SNMPv2, and CMIP – The Pratical Guide to Network–Management Standards*. Addison–Wesley, 1993.

[39] S. Waldbusser. Remote network monitoring management information base. Request for Comments (Experimental) RFC 1757, Internet Engineering Task Force, February 1995.

Acknowledgements

First of all I would like to thank Paul Christ[1], Director of the Department for BelWue Development and Communication Systems at the Computer Center of the University of Stuttgart, for supervising me during my work on this project. Without his support and guidance, the realisation would not have been possible.

Thanks go also to Wilfried Milow[2], who organized and installed the OC3MON–hardware I used for my experiments, as well as to Robert Stoy[3], who was always able to answer my questions related to ATM and who installed the optical splitters of OC3MON in the departments ATM network.

Dr. Nevil Brownlee[4], Director for Technology Development in the University of Aucklands (New Zealand) Information Technology Systems and Services Department, helped me a lot with the numerous questions I had when implementing the Java applet. I'd also like to thank Nevil for allowing me to present this work at the 39th IETF conference in Munich.

Henry Pijffers[5] helped me by providing a Java algorithm to decode BER encoded SNMP sequences.

[1] `paul.christ@rus.uni-stuttgart.de`

[2] `wmilow@str.daimler-benz.com`

[3] `robert.stoy@rus.uni-stuttgart.de`

[4] `n.brownlee@auckland.ac.nz`

[5] `pijffers@cs.utwente.nl`